PhD-FSTC-2022-135
The Faculty of Science, Technology and Communication

DISSERTATION

Defence held on 25/11/2022 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG EN INFORMATIQUE

by

Fitash UL HAQ
Born on 24th November 1993 in Jhelum (Pakistan)

SCALABLE AND PRACTICAL AUTOMATED TESTING OF DEEP LEARNING MODELS AND SYSTEMS

DISSERTATION DEFENSE COMMITTEE

Dr. Lionel Briand, Dissertation Supervisor
Professor, University of Luxembourg

Dr. Djamila Aouada, Chair
Assistant Professor, University of Luxembourg

Dr. Fabrizio Pastore, Vice Chairman
Associate Professor, University of Luxembourg

Dr. Paolo Tonella, Member
Professor, Università della Svizzera Italiana (USI)

Dr. Alessio Gambi, Member
Professor, IMC University of Applied Science Krems

Acknowledgement

Primarily, I would like to thank my supervisor, Professor Lionel Briand, for his continuous guidance and dedicated efforts in making me the researcher I am today. I am honoured to have the opportunity to work under his supervision.

I would also like to thank Dr. Donghwan Shin for his valuable feedback, discussion, and guidance in both professional and personal matters.

I am grateful to Prof. Shiva Nejati for advising me during the initial years of my PhD.

I would like to thank IEE S.A., and most particularly Dr. Thomas Stifter, for their support and insightful discussions during the PhD, as well as providing access to their case studies and simulators.

Last but not least, I would like to thank my family, friends, and colleagues for their continuous support throughout my PhD.

Fitash UL HAQ
University of Luxembourg
November 2022

Abstract

With the recent advances of Deep Neural Networks (DNNs) in real-world applications, such as Automated Driving Systems (ADS) for self-driving cars, ensuring the reliability and safety of such DNN-Enabled Systems (DES) emerges as a fundamental topic in software testing. Automatically generating new and diverse test data that lead to safety violations of DES presents the following challenges: (1) there can be many safety requirements to be considered at the same time, (2) running a high-fidelity simulator is often very computationally intensive, (3) the space of all possible test data that may trigger safety violations is too large to be exhaustively explored, (4) depending upon the accuracy of the DES under test, it may be infeasible to find a scenario causing violations for some requirements, and (5) DNNs are often developed by a third party, who does not provide access to internal information of the DNNs.

In this dissertation, in collaboration with IEE sensing, we address the aforementioned challenges by providing scalable and practical automated solutions for testing Deep Learning (DL) models and systems. Specifically, we present the following in the dissertation.

1. We conduct an empirical study to compare offline testing and online testing in the context of Automated Driving Systems (ADS). We also investigate whether simulator-generated data can be used in lieu of real-world data. Furthermore, we investigate whether offline testing results can be used to help reduce the cost of online testing.

2. We propose an approach to generate test data using many-objective search algorithms tailored for test suite generation to generate test data for DNN with many outputs. We also demonstrate a way to learn conditions that cause the DNN to mispredict the outputs.

3. In order to reduce the number of computationally expensive simulations, we propose an automated approach, SAMOTA, to generate data for DNN-enabled automated driving systems, using many-objective search and surrogate-assisted optimisation.

4. The environmental conditions (e.g., weather, lighting) often stay the same during a simulation, which can limit the scope of testing. To address this limitation, we present an automated approach, MORLAT, to dynamically interact with the environment during simulation. MORLAT relies on reinforcement learning and many-objective optimisation.

   We evaluate our approaches using state-of-the-art deep neural networks and systems. The results show that our approaches perform statistically better than the alternatives.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Neural Networks (DNNs) have been widely used in many applications, such as object detection [122], object classification [70], and speech recognition [27]. With the recent advances in Deep Learning (DL), DNNs are increasingly applied to safety-critical software systems, such as Automated Driving Systems (ADS). Examples of ADS systems include driver drowsiness detection systems, lane-keeping systems, and end-to-end autonomous driving systems. The goal of an ADS is to satisfy requirements in terms of both functional (e.g., reaching the destination in a given time) and safety (e.g., not colliding with other vehicles).

It is essential to test such DNN-Enabled Systems (DES) to ensure the safety and reliability of the system. Failures of DES can have fatal results. For example, in 2016, the autopilot in the Tesla Model S did not recognise a truck on the road and crashed into it, which resulted in the death of the Tesla driver [71]. However, real-world testing for DNN-enabled systems is time-consuming, expensive, and dangerous. Furthermore, it is very difficult to set certain conditions in the operational environment. For example, setting the weather condition to be rainy or cloudy in a controlled environment using industrial-grade sprinklers and cloud generators requires a big budget.

Due to the limitations of real-world testing, DES testing is often done using high-fidelity simulators such as CARLA [30] and LGSVL [105]. Simulator-based data generation is cheap, fast, and more diverse as compared to real-world data. High-fidelity simulators allow engineers to specify and execute driving scenarios capturing various road traffic situations, different pedestrian-to-vehicle and vehicle-to-vehicle interactions, different road typologies, weather conditions, and infrastructures. The simulators also simulate the sensors mounted on the ego vehicle, such as camera sensors, LIDAR, and RADAR. They allow DES to drive the vehicle by applying driving commands (i.e., throttle, steering, and braking) in the simulated environment.

However, automatically generating new and diverse test data for testing deep learning models and systems that lead to requirement violations entails several challenges:

(i) There can be many requirements, often independent from each other, to be considered at the same time. For example, the requirement for keeping the ego vehicle in the center of the lane is

independent from the requirement that the ego vehicle should not collide with other vehicles.

(ii) Running a high-fidelity simulator to check violations is typically computationally intensive; the higher the fidelity of a simulator, the more time it takes to simulate, which directly impacts the cost of testing.

(iii) The space of all possible test data that may trigger violations is too large to be exhaustively explored because there can be many options for different attributes. for example weather conditions can be *rainy*, *cloudy*, *foggy* or *sunny*.

(iv) Depending upon the accuracy of the system under test, it may be infeasible to find a scenario causing violations for some requirements. Considering a limited time budget, if such infeasibility is observed at run time, it is essential to dynamically and efficiently distribute computation resources to the other requirements.

(v) DNNs are often developed by a third party, with expertise in machine learning, who does not provide access to internal information of the DNNs.

In this dissertation, we address the aforementioned challenges of testing deep learning models and systems. We start with investigating whether simulator-generated data can be used in-lieu of real-world data. We further define and compare two general modes of testing for DNNs: offline and online testing. We then present an offline testing approach based on many-objective optimisation to generate test data for DNNs with many outputs. We also present an online testing approach that leverages surrogate-assisted optimisation to address the high computational cost of simulation in online testing. Furthermore, we present another online testing approach that uses Reinforcement Learning (RL) to address the limitation of existing online testing approaches that do not change the dynamic elements of the environment during the simulation. The work in this dissertation has been done in collaboration with IEE sensing, who provides advanced sensing solutions to the automotive industry [1].

Throughout this dissertation, we use different simulators to test different DES. We use the PreScan [123] simulator to compare different modes of DNN testing. PreScan is a widely used commercial simulator in the automotive domain and is used by IEE to test their sensors. We use IEE-Sim, a simulator developed in-house by IEE to generate facial images, using MakeHuman and blender, with actual keypoints positions (ground truth) to test keypoints detection DNNs. We use CARLA [30] to test end-to-end DNN-enabled systems. CARLA is a widely used open source simulator designed for training, verification, and validation of automated driving systems.

1.1 Research Contributions

The research contributions of this dissertation can be summarised as follows:

- We report an empirical study to compare offline testing and online testing in the context of Automated Driving Systems (ADS). We also investigate whether simulator-generated data is a reliable substitute for real-world data for the purpose of DNN testing. Furthermore, we investigate if we can exploit offline testing results to reduce the cost of online testing by running fewer tests. Part of this work was published at the International Conference on Software Testing, Verification

and Validation (ICST) 2020 [51] and in the Empirical Software Engineering (EMSE) [53] journal. This work is presented in Chapter 3.

- We present an approach to automatically generate test data for Key-Points detection DNNs (KP-DNNs) using many-objective optimisation and simulation. Our approach uses many objective search algorithms tailored for test suite generation to find test cases that mispredict at least one keypoint. We further investigate and demonstrate a way to learn the conditions that cause severe mispredictions for individual keypoints. This work was published in the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA) 2021 [49] and is presented in Chapter 4.

- We present SAMOTA (Surrogate-Assisted Many-Objective Testing Approach), an approach to automatically and efficiently generate test data for DES testing, in online mode, by carefully combining (1) many-objective optimisation, to effectively achieve many independent objectives (i.e., causing safety violations) within a limited time budget, and (2) surrogate-assisted optimization to efficiently search for critical test data using surrogate models that mimic the simulator. This work was published in International Conference on Software Engineering (ICSE) 2022 and was awarded an **ACM-SIGSOFT Distinguished Paper Award**. This work is presented in Chapter 5.

- We present MORLOT (Many-Objective Reinforcement Learning for Online Testing), an approach to effectively generate test data for DES testing by combining (1) reinforcement learning, to dynamically interact with the environment and (2) many-objective optimisation, to efficiently solve many independent objectives simultaneously. This work is currently under review and is presented in Chapter 6.

1.2 Dissertation Outline

- **Chapter 2** provides background on search-based testing, simulation-based testing, surrogate models, and reinforcement learning.

- **Chapter 3** presents an empirical study to compare offline testing and online testing in the context of Automated Driving Systems (ADS).

- **Chapter 4** presents our approach for testing Key-Points detection DNNs (KP-DNNs) using many-objective optimisation.

- **Chapter 5** presents SAMOTA, our approach for testing end-to-end DNN-enabled systems using many-objective optimisation and surrogate models.

- **Chapter 6** presents MORLOT, our approach for testing end-to-end DNN-enabled systems using many-objective optimisation and reinforcement learning.

Chapter 2

Background

This chapter presents background concepts used throughout the dissertation. Section 2.1 presents the basic concepts of search-based testing. Section 2.2 presents background on simulation-based testing. Section 2.3 introduces surrogate models and three widely used surrogate models and section 2.4 introduces basic concepts about reinforcement learning.

2.1 Search-based Testing

Search-based software testing (SBST) [86] uses meta-heuristic algorithms to automate software testing tasks, such as test case generation [37] and prioritization [75], for a specific system under test. The key idea is to formulate a software testing problem as an optimization problem by defining proper fitness functions. For example, EvoSuite [37] uses a search-based approach to automatically generate unit test cases for a Java program to satisfy a coverage criterion, such as branch coverage. In this case, the fitness function is defined based on the coverage achieved by unit test cases.

Recently, SBST has also been used for DNN testing. For example, AsFault [41] automatically generates virtual roads to make vision-based DNNs go out of lane, and DeepJanus [103] uses multi-objective search to generate a pair of similar test inputs that cause the DNN under test to misbehave for one test input but not for the other.

However, when the number of objectives is above three, multi-objective search algorithms, such as NSGA-II [29], do not scale well [66, 21]. This is where many-objective search algorithms come into play. For example, NSGA-III [28] is a generic many-objective search algorithm that extends NSGA-II with the idea of virtual reference points to increase the diversity of optimal solutions even when there are more than three objectives. Panichella et al. [96] proposed a Many-Objective Sorting Algorithm (MOSA), another extension of NSGA-II, that is tailored for test suite generation. In contrast to NSGA-III, MOSA aims to efficiently achieve each objective individually. To do this, MOSA has three main features: (1) it focuses search towards uncovered objectives, (2) it uses a novel preference criterion to rank solutions rather than diversifying them, and (3) it saves the best test case for each objective in an archive. MOSA has shown to outperform alternative search algorithms in the context of coverage-based unit testing for

traditional software programs [95, 96]. Recently, Abdessalem et al. [4] proposed FITEST, an extension of MOSA, to further improve the efficiency of many-objective search. The idea is to dynamically reduce the number of candidates (i.e., the population size) to be considered by focusing on the uncovered objectives only. Hence, FITEST's population size decreases as more objectives are achieved, whereas MOSA's population size is fixed throughout the search. Notice that MOSA and FITEST are carefully designed for search-based test suite generation when the number of objectives is above three.

2.2 Simulation-based Testing

Testing cyber-physical, safety-critical systems can be done in two ways: (1) in a real-world environment and (2) in a virtual environment, relying on simulators. In the former case, software is deployed in the real-world environment, a form of testing which is often expensive and dangerous. The latter case, referred to as simulation-based testing, can raise concerns about simulation's fidelity but has two major benefits. First, simulation-based testing offers controllability, meaning that test drivers can control static and dynamic features of the simulation (e.g., topology of roads, face features), and can thus automatically cover a wide range of scenarios. Second, because we can control simulation parameters and can get information from the simulator, we know the ground truth and can thus apply search-based solutions to perform safe and fully automated testing of safety-critical software [2, 5, 12].

In many cyber-physical fields, simulation models are developed before implementing actual products. These simulation models help engineers in various activities, such as early verification and validation of the system under test [87, 56]. In such contexts, simulated-based testing is highly recommended as it is economical, faster, safer and flexible [56].

2.3 Surrogate Models

Evolutionary Algorithms (EAs) have been successfully applied to many complex engineering problems [36]. However, the fitness function evaluation of such complex problems often involves computationally expensive simulations or calculations [62]. To address this issue, many researchers have investigated *surrogate models* that can replace the computationally expensive function evaluations with much less expensive approximations. Among them, we briefly introduce the most widely used surrogate model types, i.e., Kriging [118], polynomial regression [119], radial basis function networks [16], and their ensemble [45].

2.3.1 Kriging

Kriging (also known as gaussian process regression) is one of the most widely used surrogate models since the 1970s. Similar to regression analysis, it predicts the value of a function as a combination of linear functions using a stochastic process. While it provides the error value for each prediction, it is relatively more time-consuming for training than other surrogate models.

2.3.2 Polynomial Regression

As a form of statistical regression analysis, polynomial regression models the relationship between the independent variable x and the dependent variable y in the form of nth degree polynomial in x. Though it

is simple and intuitive, the existence of a few outliers can severely distort the approximation in nonlinear problems [90].

2.3.3 Radial Basis Function Network

A radial basis function network is an artificial neural network that consists of three layers: input, hidden, and output layers. Radial basis functions are used as activation functions, and the output of the network is a weighted sum of radial basis functions. It is known to provide both computational efficiency and reasonable training accuracy [78].

2.3.4 Ensemble

Though many surrogate models have been studied, there is no single surrogate model that consistently performs well for all problems [60]. To mitigate the issue, the ensemble of different surrogate models can be considered. Specifically, given an ensemble model $\mathbf{m}$ composed of member models $m_1, \ldots, m_k$, the final output of $\mathbf{m}$ for an input x, denoted with $\hat{y}_{\mathbf{m}}(x)$, is the weighted sum of all k member outputs as $\hat{y}_{\mathbf{m}}(x) = \sum_{i=1}^{k} w_i \times \hat{y}_{m_i}(x)$ where $\hat{y}_{m_i}(x)$ is the output of m_i for x and w_i is a weight for $\hat{y}_{m_i}(x)$. Following Goel et al. [45], w_i is defined as $w_i = \frac{(\sum_{p=1}^{k} e_p) - e_i}{(k-1)\sum_{i=1}^{k} e_i}$ where e_i is the training error of m_i.

Another advantage of using ensemble models is that we can easily compute the *uncertainty* of a prediction of an ensemble model. Specifically, the uncertainty of $\hat{y}_{\mathbf{m}}(x)$, denoted with $\delta_{\mathbf{m}}(x)$, is defined as $\delta_{\mathbf{m}}(x) = max_{i,p}(|\hat{y}_{m_i}(x) - \hat{y}_{m_p}(x)|)$ for $i, p \in \{1, 2, \ldots, k\}$ and $i \neq p$. In other words, the uncertainty is calculated using the maximum difference between the outputs of the k member models.

2.4 Reinforcement Learning

Reinforcement Learning (RL) is about learning how to perform a sequence of actions to achieve a goal by iterating trials and errors to learn the best action for a given state [121].

In particular, RL involves the interaction between an RL agent (i.e., the learner) and its surrounding environment, which is formalized by a Markov Decision Process (MDP). At each time step j, the RL agent observes the environment's state s_j and takes an action a_j based on its own policy π (i.e., the mapping between states and actions). At the next time step $j+1$, the agent first gets a reward w_{j+1} indicating how well taking a_j in s_j helped in achieving the goal, updates π based on w_{j+1}, and then continues to interact with its environment. From the interactions (trials and errors), the agent is expected to learn the unknown optimal policy π^* that can select the best action maximizing the expected sum of future rewards in any state.

An important assumption underlying MDP is that states satisfy the *Markov property*: states captures information about all aspects of the past agent–environment interactions that make a difference for the future [121]. In other words, w_{j+1} and s_{j+1} depend only on s_j and a_j, and are independent from the previous states $s_{j-1}, s_{j-2}, \ldots, s_1$ and actions $a_{j-1}, a_{j-2}, \ldots, a_1$. This assumption allows the RL agent to take an action by considering only the current state, as opposed to all past states (and actions).

In general, there are two types of RL methods: (1) *tabular*-based and (2) *approximation*-based. Tabular-based methods [136, 108] use tables or arrays to store the expected sum of future rewards for each state. Though they are applicable only if state and action spaces are small enough to be represented in tables or arrays, they can often find exactly the optimal policy [121]. Discretization can be used to

control the size of state and action spaces, especially when states and actions are continuous. It is essential to apply the right degree of discretization since coarse-grained discretization may make it impossible for the agent to distinguish between states that require different actions, resulting in significant loss of information. When the state space is enormous and cannot be easily discretized without significant information loss, approximation-based methods [8] can be used where the expected sum of future rewards for a newly discovered state can be approximated based on known states (often with the help of state abstraction when they are too complex to directly compare). While they can address complex problems in very large state spaces, they can only provide approximate solutions.

One of the most commonly used tabular-based reinforcement learning algorithms is Q-learning due to its simplicity and guaranteed convergence to an optimal policy [136]. It stores and iteratively updates the expected sum of future rewards for each state-action pair in a table (a.k.a., Q-table) while going through trials and errors. A properly updated Q-table can therefore tell what is the best action to choose in a given state. A more detailed explanation, including how to update a Q-table to ensure the convergence, can be found in Sutton and Barto [121].

Chapter 3

Modes of Testing: Online Testing vs Offline testing

In this chapter, we perform an empirical study to compare offline testing and online testing in the context of Automated Driving Systems (ADS). In offline testing, DNNs are tested as individual units based on test datasets obtained without involving the DNNs under test, while in online testing, DNNs are embedded into a specific application environment and tested in a closed-loop mode in interaction with the application environment. We aim to answer the following research question: *How do offline and online testing results differ and complement each other?* To answer this question, we used open-source DNN models developed to automate steering functions of self-driving vehicles [128]. To enable online testing of these DNNs, we integrated them into a powerful, high-fidelity physics-based simulator of self-driving cars [123]. The simulator allows us to specify and execute scenarios capturing various road traffic situations, different pedestrian-to-vehicle and vehicle-to-vehicle interactions, and different road topologies, weather conditions and infrastructures. As a result, in our study offline and online testing approaches were compared with respect to the data generated automatically using a simulator. To ensure that this aspect does not impact the validity of our comparison, we investigate the following research question as a pre-requisite of the above question: *Can we use simulator-generated data as a reliable substitute to real-world data for the purpose of DNN testing?*

While the above research questions provide insights on the relationship between offline and online testing results, it is still unclear how we can use offline and online testing together in practice such that we can minimize cost and maximize the effectiveness of testing DNNs. As the ML testing workflow in Figure 3.1 suggests, offline testing precedes online testing and given that offline testing is considerably less expensive than online testing, it is beneficial if we can exploit offline testing results to reduce the cost of online testing by running fewer tests. To explore this, we investigate the following research question to determine *if offline testing results can be used to help reduce the cost of online testing?* Our goal is to identify whether we can characterize the test scenarios (conditions) where offline and online testing results are the same with high probability. To do so, we propose a novel heuristic approach to infer such

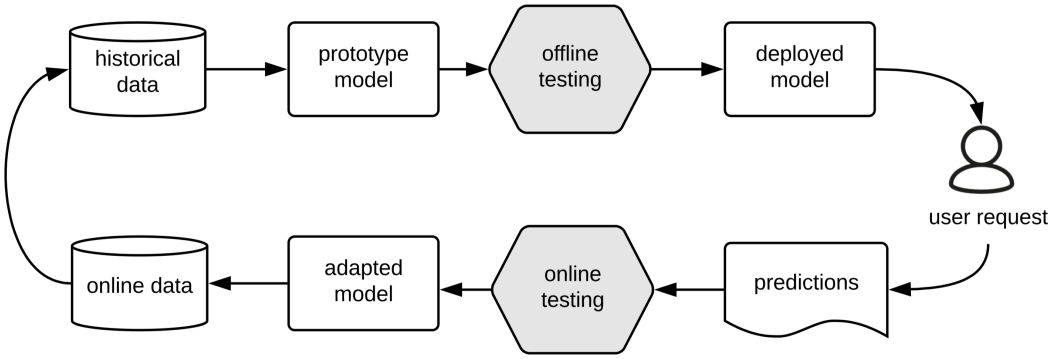


Figure 3.1: Idealized Workflow of ML testing [147]

conditions from limited number of offline and online testing data in an efficient and effective way.

The contributions of this chapter are summarized below:

1. We show that we can use simulator-generated datasets in lieu of real-life datasets for testing DNNs in our application context. Our comparison between online and offline testing using such datasets show that offline and online testing results frequently differ, and specifically, offline testing results are often not able to find faulty behaviors due to the lack of error accumulation over time. As a result, many safety violations identified by online testing could not be identified by offline testing as they did not cause large prediction errors. However, all the large prediction errors generated by offline testing led to severe safety violations detectable by online testing.

2. We provide a three-step approach to infer (learn) conditions characterizing agreement and disagreement between offline and online testing results while minimizing the amount of the data required to infer the conditions and maximizing the statistical confidence of the results.

3. We were not able to infer any conditions that can characterize agreement between offline and online testing results with a probability higher than 71%. This means that, in general, we cannot exploit offline testing results to reduce the cost of online testing in practice.

The rest of the chapter is organized as follows: Section 3.1 introduces offline and online testing, describes our proposed domain model that is used to configure simulation scenarios for automated driving systems, and formalizes the main concepts in offline and online testing used in our experiments. Section 3.2 reports on the empirical evaluation. Section 3.3 provides discussion on online testing using simulators, offline testing vs online testing and open challenges. Section 3.4 surveys the existing research on online and offline testing for automated driving systems. Section 3.5 concludes the chapter.

3.1 Offline and Online testing frameworks

This section provides the basic concepts that will be used throughout the chapter.

3.1.1 DNNs in ADS

Depending on the ADS design, DNNs may be used in two ways to automate the driving task of a vehicle: One design approach is to incorporate DNNs into the ADS perception layer, primarily to do

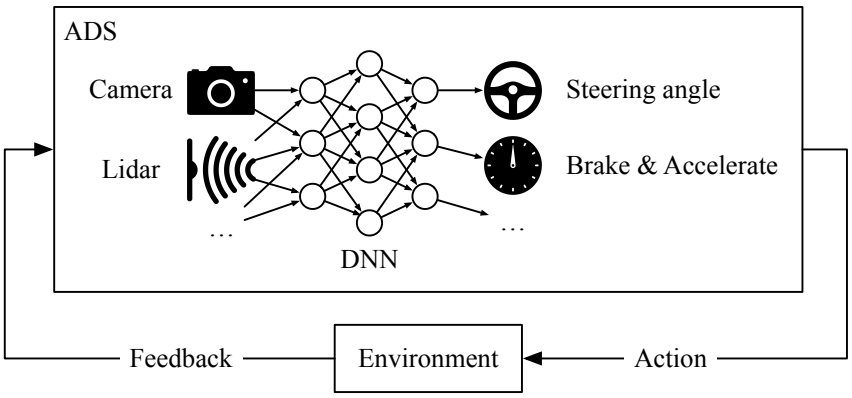


Figure 3.2: Overview of DNN-based ADS

semantic segmentation [43], i.e., to classify and label each and every pixel in a given image. The ADS software controller then decides what commands should be issued to the vehicle's actuators based on the classification results produced by the DNN [101]. An alternative design approach is to use DNNs to perform the *end-to-end* control of a vehicle [128] (e.g., Figure 3.2). In this case, DNNs directly generate the commands to be sent to the vehicle's actuators after processing images received from cameras. Our approach to compare offline and online testing of DNNs is applicable to both ADS designs. In the comparison provided in this chapter, however, we use DNN models automating the end-to-end control of the steering function since these models are publicly available online and have been extensively used in recent studies on DNN testing [125, 148, 81, 64]. In particular, we investigate the DNN models from the Udacity self-driving challenge as our study subjects [128]. We refer to this class of DNNs as ADS-DNNs in the remainder of the chapter. Specifically, an ADS-DNN receives as input images from a front-facing camera mounted on a vehicle, and generates a steering angle command for the vehicle.

3.1.2 Test Data Sources

We identify two sources for generating test data for testing ADS-DNNs: (1) real-life driving and (2) driving simulator.

For our ADS-DNN models, a *real-life dataset* is a video or a sequence of images captured by a camera mounted on a vehicle's dashboard while the vehicle is being driven by a human driver. The steering angle of the vehicle applied by the human driver is recorded for the duration of the video and each image (frame) of the video in this sequence is labelled by its corresponding steering angle. This yields a sequence of manually labelled images to be used for testing DNNs. There are, however, some drawbacks with test datasets captured from real-life [63]. Specifically, data generation is expensive, time consuming and lacks diversity. The latter issue is particularly critical since driving scenes, driving habits, as well as objects, infrastructures and roads in driving scenes, can vary widely across countries, continents, climates, seasons, day times, and even drivers.

Another source of test data generation is to use simulators to automatically generate videos capturing various driving scenarios. There are increasingly more high-fidelity and advanced physics-based simulators for self-driving vehicles fostered by the needs of the automotive industry, which increasingly relies on simulators to improve their testing and verification practices. There are several examples of commercial ADS simulators (e.g., PreScan [123] and Pro-SiVIC [35]) and a number of open source ones (e.g., CARLA [31] and LGSVL [106]). These simulators incorporate dynamic models of vehicles (including

vehicles' actuators, sensors and cameras) and humans as well as various environment aspects (e.g., weather conditions, different road types, different infrastructures). The simulators are highly configurable and can be used to generate desired driving scenarios. In our work, we use the PreScan simulator to generate test data for ADS-DNNs. PreScan is a widely-used, high-fidelity commercial ADS simulator in the automotive domain and has been used by our industrial partner. In Section 3.1.3, we present the domain model that define the inputs used to configure the simulator, and describe how we automatically generate scenarios that can be used to test ADS-DNNs. Similar to real-life videos, the videos generated by our simulator are sequences of labelled images such that each image is labelled by a steering angle. In contrast to real-life videos, the steering angles generated by the simulator are automatically computed based on the road trajectory as opposed to being generated by a human driver.

The simulator-generated test datasets are cheaper and faster to produce compared to real-life ones. In addition, depending on how advanced and comprehensive the simulator is, we can achieve a higher-level of diversity in the simulator-generated datasets by controlling and varying the objects, roads, weather, and other various features. However, it is not yet clear whether simulator-generated images can be used in lieu of real images since the latter may have higher resolution, showing more natural texture, and look more realistic. In this chapter, we conduct an empirical study in Section 3.2 to investigate *if we can use simulator-generated images as a reliable alternative to real images for testing ADS-DNNs*.

3.1.3 Domain Model

Figure 3.3 shows the domain model capturing the test input space of ADS-DNNs. To develop the domain model, we relied on two sources of information: (1) the properties that we observed in the real-world ADS-DNN test datasets (i.e., the Udacity testing datasets [129]) and (2) the configurable parameters of our simulator.

In total, we identified four main objects, i.e., *Road*, *Vehicle*, *Weather*, and *Environment*, and 32 attributes characterizing them, such as *Road.type*, *Vehicle.speed*, *Weather.type*, and *Environment.buildings*. Each attribute has a specific data type; for example, the *Weather.type* attribute is an enumeration type, having three different weather values (i.e., *Snowy*, *Sunny*, and *Rainy*) as shown in the definition of *Weather.Type* in Figure 3.3. This means that only one of the three values can be assigned to *Weather.type*. Note that, to illustrate the lower diversity in real-world datasets, the attributes and their values that are observed in the real world are highlighted in bold. For example, only the *Sunny* weather is observed in the real-world test datasets.

In addition to objects and attributes, our domain model includes some constraints describing valid value assignments to the attributes. These constraints mostly capture the physical limitations and traffic rules that apply to our objects. For example, the vehicle speed cannot be higher than 20km/h on steep curved roads. Constraints may also be used to capture dependencies between attributes that cannot be specified through the relationships between domain model objects. For example, we define a constraint to indicate that *Weather.condition* can only take a value when *Weather.type* is either *Snowy* or *Rainy*. That is, for *Sunny* we do not need to specify any weather condition. We have specified these constraints in the Object Constraint Language (OCL) [47]. The complete OCL constraints are available in the supporting materials [50].

To produce a simulation scenario (or test scenario) for an ADS-DNN, we instantiate our domain model in Figure 3.3 by assigning concrete values to the attributes of our domain model such that its OCL

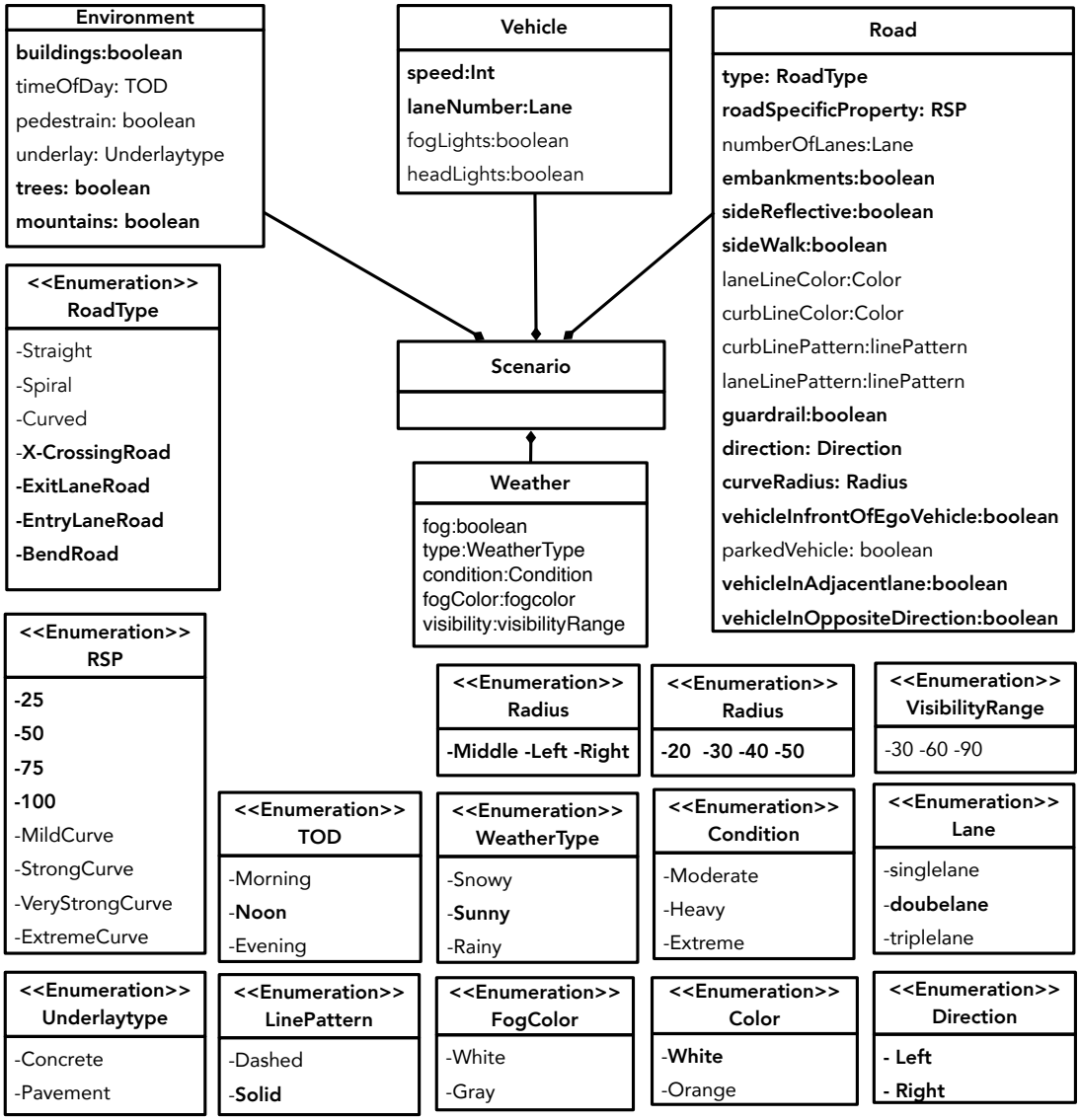


Figure 3.3: Complete domain model for scenario generation. The attributes and values that are observed in the real-world test datasets are highlight in bold.

constraints are satisfied. Specifically, we can represent each test scenario as a vector $\mathbf{s} = \langle v_1, v_2, \ldots, v_{32} \rangle$ where v_i is the value assigned to the ith attribute of our domain model (recall that it contains 32 attributes). We can then initialize the simulator based on the test scenario vectors. The simulator will then generate, for each of the mobile objects defined in a scenario, namely the ego and secondary vehicles and pedestrians, a trajectory vector of the path of that object (i.e., a vector of values indicating the positions and speeds of the mobile object over time). The length of the trajectory vector is determined by the duration of the simulation. The position values are computed based the characteristics of the static objects specified by the initial configuration, such as roads and sidewalks, as well as the speed of the mobile objects.

3.1.4 Offline Testing

Figure 3.4 represents an overview of offline DNN testing in the ADS context. Briefly, offline testing verifies the DNN using historical data consisting of sequences of images captured from real-life camera

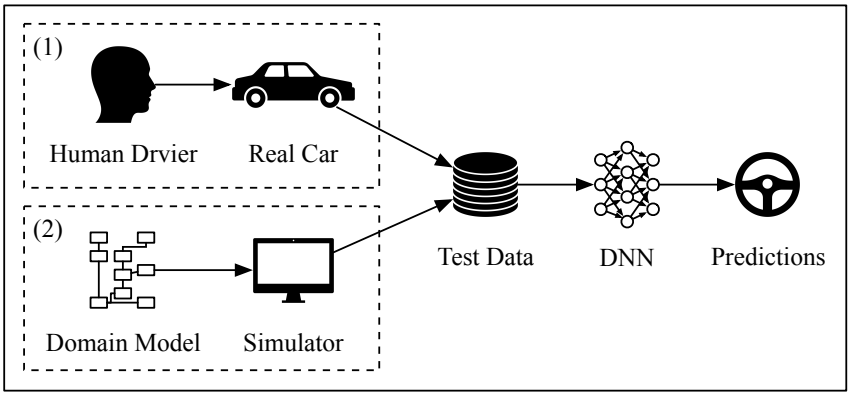


Figure 3.4: Offline testing using (1) real-world and (2) simulator-generated data

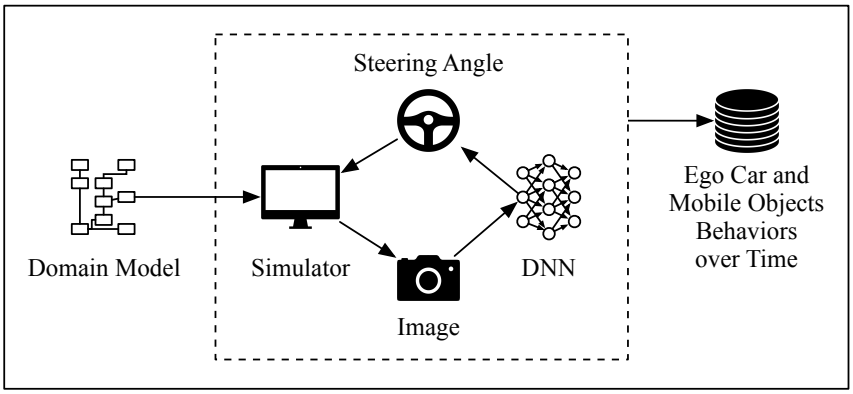


Figure 3.5: Online testing of ADS-DNNs using simulators

or based on a camera model of a simulator. In either case, the images are labelled with steering angles. Offline testing measures the *prediction errors* of the DNN to evaluate test results.

More specifically, let $\mathbf{r}$ be a real-life test dataset composed of a sequence of tuples $\langle (i_1^r, \theta_1^r), (i_2^r, \theta_2^r), \ldots, (i_n^r, \theta_n^r) \rangle$. For $j = 1, \ldots, n$, each tuple (i_j^r, θ_j^r) of $\mathbf{r}$ consists of an image i_j^r and a steering angle θ_j^r label. A DNN d, when provided with a sequence $\langle i_1^r, i_2^r, \ldots, i_n^r \rangle$ of the images of $\mathbf{r}$, returns a sequence $\langle \hat{\theta}_1^r, \hat{\theta}_2^r, \ldots, \hat{\theta}_n^r \rangle$ of predicted steering angles. The prediction error of d for $\mathbf{r}$ is, then, computed using two well-known metrics, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), defined below:

$$MAE(d, \mathbf{r}) = \frac{\sum_{i=1}^{n} |\theta_i^r - \hat{\theta}_i^r|}{n}$$

$$RMSE(d, \mathbf{r}) = \sqrt{\frac{\sum_{i=1}^{n} (\theta_i^r - \hat{\theta}_i^r)^2}{n}}$$

To generate a test dataset using a simulator, we provide the simulator with an initial configuration of a scenario as defined in Section 3.1.3. We denote the offline test dataset generated by a simulator for a scenario $\mathbf{s}$ by $sim(\mathbf{s}) = \langle (i_1^s, \theta_1^s), (i_2^s, \theta_2^s), \ldots, (i_n^s, \theta_n^s) \rangle$. The prediction error of d for $sim(\mathbf{s})$ is calculated by the MAE and RMSE metrics in the same way as $MAE(d, \mathbf{r})$ and $RMSE(d, \mathbf{r})$, replacing $\mathbf{r}$ with $sim(\mathbf{s})$.

3.1.5 Online Testing

Figure 3.5 provides an overview of online testing of DNNs in the ADS context. In contrast to offline testing, DNNs are embedded into a driving environment, often in a simulator due to the cost and risk of real-world testing as we described in Section 3.1.2. DNNs then receive images generated by the simulator, and their outputs are directly sent to the (ego) vehicle models of the simulator. With online testing, we can evaluate how predictions generated by an ADS-DNN, for an image generated at time t in a scenario, impact the images to be generated at the time steps after t. In addition to the steering angle outputs directly generated by the ADS-DNN, we obtain the trajectory outputs of the ego vehicle, which enable us to determine whether the vehicle is able to stay in its lane.

More specifically, we embed a DNN d into a simulator and run the simulator. For each (initial configuration of a) scenario, we execute the simulator for a time duration T. The simulator generates the trajectories of mobile objects as well as images taken from the front-facing camera of an ego vehicle at regular time steps t_δ, generating outputs as vectors of size $m = \lfloor \frac{T}{t_\delta} \rfloor$. Each simulator output and image takes an index between 1 to m. We refer to the indices as simulation time steps. At each time step j, the simulator generates an image i^s_j to be sent to d as input, and d predicts a steering angle $\hat{\theta}^s_j$ which is sent to the simulator. The status of the ego vehicle is then updated in the next time step $j+1$ (i.e., the time duration it takes to update the vehicle is t_δ) before the next image i^s_{j+1} is generated. In addition to images, the simulator generates the position of the ego vehicle over time. Recall that the main function of our DNN is automated lane keeping. This function is violated when the ego vehicle departs from its lane. To measure the lane departure degree, we use the Maximum Distance from Center of Lane (MDCL) metric for the ego vehicle to determine if a safety violation has occurred. The value of MDCL is computed at the end of the simulation when we have the position vector of the ego vehicle over time steps, which was guided by our DNN. We cap the value of MDCL at $1.5\,\mathrm{m}$, indicating that when MDCL is $1.5\,\mathrm{m}$ or larger, the ego vehicle has already departed its lane and a safety violation has occurred. In addition, we normalize the MDCL values between 0 and 1 to make it consistent with MAE or RMSE.

In this chapter, we embed the ADS-DNN into PreScan by providing the former with the outputs from the camera model in input and connecting the steering angle output of the ADS-DNN to the input command of the vehicle dynamic model.

3.2 Experiments

We aim to compare offline and online testing of DNNs by answering the following research questions:

RQ1: *Can we use simulator-generated data as a reliable alternative source to real-world data?* Recall the two sources for generating test data as described in Section 3.1.2. While simulator-generated test data is cheaper and faster and is more amenable to input diversification compared to real-life test data, the texture and resolution of real-life data look more natural and realistic compared to the simulator-generated data. In RQ1, we aim to investigate whether, or not, such differences lead to significant inaccuracies in predictions of the DNN under test in offline testing. The answer to this question will determine if we can rely on simulator-generated data for testing DNNs in either offline or online testing modes.

RQ2: *How frequently do offline and online testing results differ and do they complement each other?* RQ2 is one of the main research questions we want to answer in this chapter. We want to know how the results obtained by testing a DNN in isolation, irrespective of a particular application context, compare

with the results obtained by embedding a DNN into a specific application environment. The answer to this question will help engineers and researchers better understand the applications and limitations of each testing mode, and how they could possibly be combined.

RQ3: *Can offline testing results be used to help reduce the cost of online testing?* In other words, can we focus online testing on situations where it is needed, i.e., on situations where offline and online testing are in disagreement? With RQ3, we investigate whether any offline testing results can be lifted to online testing to help reduce the amount of online testing that we need to do. Our goal is to determine whether we can characterize the test scenarios where offline and online testing behave the same in terms of our domain model elements. This provides the conditions under which offline testing is sufficient, thus avoiding online testing, which is much more expensive.

3.2.1 Experimental Subjects

We use three publicly-available, pre-trained DNN-based steering angle prediction models, i.e., Autumn [9], Chauffeur [18], and Komanda [67], that have been widely used in previous work to evaluate various DNN testing approaches [125, 148, 64].

Autumn consists of an image preprocessing module implemented using OpenCV to compute the optical flow of raw images, and a Convolutional Neural Network (CNN) implemented using Tensorflow and Keras to predict steering angles. Autumn improved performance by using cropped images from the bottom half of the entire images. Chauffeur consists of one CNN that extracts the features from raw images and a Recurrent Neural Network (RNN) that predicts steering angles from the previous 100 consecutive images with the aid of a LSTM (Long Short-Term Memory) module. Similar to Autumn, Chauffeur uses cropped images, and is also implemented with Tensorflow and Keras. Komanda consists of one CNN followed by one RNN with LSTM, implemented by Tensorflow, similar to Chauffeur. However, the underlying CNN of Komanda has one more dimension than Chauffeur that is in charge of learning spatiotemporal features. Further, unlike Autumn and Chauffeur, Komanda uses full images to predict steering angles.

The models are developed using the Udacity dataset [129], which contains 33808 images for training and 5614 images for testing. The images are sequences of frames of two separate videos, one for training and one for testing, recorded by a dashboard camera with 20 Frame-Per-Second (FPS). The dataset also provides, for each image, the actual steering angle produced by a human driver while the videos were recorded. A positive (+) steering angle represents turning right, a negative (-) steering angle represents turning left, and a zero angle represents staying on a straight line. The steering angle values are normalized (i.e., they are between -1 and $+1$) where a $+1$ steering angle value indicates $25°$, and a -1 steering angle value indicates $-25°$[1]. Figure 3.6 shows the actual steering angle values for the sequence of 5614 images in the test dataset. We note that the order of images in the training and test datasets matters and is accounted for when applying the DNN models. As shown in the figure, the steering angles issued by the driver vary considerably over time. The large steering angle values (more than $3°$) indicate actual road curves, while the smaller fluctuations are due to the natural behavior of the human driver even when the vehicle drives on a straight road.

[1]This is how Tian et al. [125] have interpreted the steering angle values provided along with the Udacity dataset, and we follow their interpretation. We were not able to find any explicit information about the measurement unit of these values anywhere else.

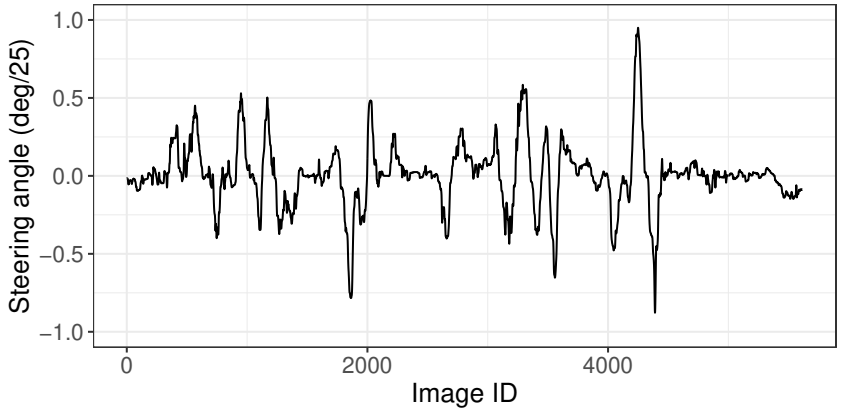


Figure 3.6: Actual steering angles for the 5614 real-world images used for testing

Table 3.1: Accuracies of the subject DNN-based models

Model	Reported RMSE	Our RMSE	Our MAE
Autumn	Not Presented	0.049	0.034
Chauffeur	0.058	0.092	0.055
Komanda	0.048	0.058	0.039

Table 3.1 shows the RMSE and MAE values of the models we obtained for the Udacity test dataset, as well as the RMSE values reported by the Udacity website [128] [2]. The differences are attributed to challenges regarding reproducibility, a well-known problem for state-of-the-art deep learning methods [100] because they involve many parameters and details whose variations may lead to different results. Specifically, even though we tried to carefully follow the same settings and parameters as those suggested on the Udacity website, as shown in Table 3.1, the RMSE and MAE values that we computed differed from those reported by Udacity. We believe these differences are due to the versions of python and other required libraries (e.g., tensorflow, keras, and scipy). The precise version information for all these were not reported by Udacity. Nevertheless, for all of our experiments, we consistently used the most stable versions of python and the libraries that were compatible with one another. In other words, irrespective of differences in the RMSE values between the reported and our in Table 3.1, all of our experiments are internally consistent. To enable replication of our work, we have made our detailed configurations (e.g., python and auxiliary library versions), together with supporting materials, available online [50].

While MAE and RMSE are two of the most common metrics used to measure prediction errors for learning models with continuous variable outputs, we mainly use MAE throughout this chapter because, in contrast to RMSE, MAE values can be directly interpreted in terms of individual steering angle values. For example, $MAE(d, \mathbf{r}) = 1$ means that the average prediction error of d for the images in $\mathbf{r}$ is 1 (25°). Since MAE is a more intuitive metric for our purpose, we will only report MAE values in the remainder of this chapter.

[2] Autumn's RMSE is not presented in the final leaderboard.

3.2.2 RQ1: Comparing Offline Testing Results for Real-life Data and Simulator-generated Data

Setup

We aim to generate simulator-generated datasets closely mimicking the Udacity real-life test dataset and verify whether the prediction errors obtained by applying DNNs to the simulator-generated datasets are comparable with those obtained for their corresponding real-life ones. As explained in Section 3.2.1, our real-life test dataset is a sequence of 5614 images labelled with their corresponding actual steering angles. If we could precisely extract the properties of the environment and the dynamics of the ego vehicle from the real-life datasets, in terms of initial configuration parameters of the simulator, we could perhaps generate simulated data closely resembling the real-life videos. However, extracting information from such video images to generate inputs of a simulator is not possible.

Instead, we propose a two-step heuristic approach to replicate the real-life dataset using our simulator. Basically, we steer the simulator to generate a sequence of images similar to the images in the real-life dataset such that the steering angles generated by the simulator are close to the steering angle labels in the real-life dataset.

In the first step, we observe the test dataset and manually identify the information in the images that correspond to some attribute values in our domain model described in Section 3.1.3. We then create a "restricted" domain model by fixing the attribute values in our domain model to the values we observed in the Udacity test dataset. This enables us to steer the simulator to resemble the characteristics of the images in the test dataset to the extent possible. Our restricted domain model includes the attributes and its values that are highlighted in bold in Figure 3.3. For example, the restricted domain model does not include weather conditions other than sunny because the test dataset has only sunny images. This guarantees that the simulator-generated images based on the restricted domain model represent sunny scenes only. Using the restricted domain model, we randomly generate a large number of scenarios yielding a large number of simulator-generated datasets.

In the second step, we aim to ensure that the datasets generated by the simulator have similar steering angle labels as the labels in the real-life dataset. To ensure this, we match the simulator-generated datasets with (sub)sequences of the Udacity test dataset such that the similarities between their steering angles are maximized. Note that steering angle is *not* a configurable attribute in our domain model, and hence, we could not force the simulator to generate data with steering angle values identical to those in the test dataset by restricting our domain model. In other words, we minimize the differences by selecting the closest simulator-generated datasets from a large pool of randomly generated ones. To do this, we define, below, the notion of "comparability" between a real-life dataset and a simulator-generated dataset in terms of steering angles.

Let S be a set of randomly generated scenarios using the restricted domain model, and let $\mathbf{r} = \langle(i_1^r, \theta_1^r), \ldots, (i_k^r, \theta_k^r)\rangle$ be the Udacity test dataset where $k = 5614$. We denote by $\mathbf{r}_{(x,l)} = \langle(i_{x+1}^r, \theta_{x+1}^r), \ldots, (i_{x+l}^r, \theta_{x+l}^r)\rangle$ a subsequence of $\mathbf{r}$ with length l starting from index $x+1$ where $x \in \{0, 1, \ldots, k\}$. For a given simulator-generated dataset $sim(\mathbf{s}) = \langle(i_1^s, \theta_1^s), \ldots, (i_n^s, \theta_n^s)\rangle$ corresponding to a scenario

$\mathbf{s} \in S$, we compute $\mathbf{r}_{(x,l)}$ using the following three conditions:

$$l = n \tag{3.1}$$

$$x = \underset{x}{\operatorname{argmin}} \sum_{j=1}^{l} |\theta_j^s - \theta_{x+j}^r| \tag{3.2}$$

$$\frac{\sum_{j=1}^{l} |\theta_j^s - \theta_{x+j}^r|}{l} \leq \epsilon \tag{3.3}$$

where $\operatorname{argmin}_x f(x)$ returns[3] x minimizing $f(x)$, and ϵ is a small threshold on the average steering angle difference between $sim(\mathbf{s})$ and $\mathbf{r}_{(x,l)}$. We say datasets $sim(\mathbf{s})$ and $\mathbf{r}_{(x,l)}$ are *comparable* if and only if $\mathbf{r}_{(x,l)}$ satisfies the three above conditions (i.e., 3.1, 3.2 and 3.3).

Given the above formalization, our approach to replicate the real-life dataset $\mathbf{r}$ using our simulator can be summarized as follows: In the first step, we randomly generate a set of many scenarios S based on the reduced domain model. In the second step, for every scenario $\mathbf{s} \in S$, we identify a subsequence $\mathbf{r}_{(x,l)}$ from $\mathbf{r}$ such that $sim(\mathbf{s})$ and $\mathbf{r}_{(x,l)}$ are comparable.

If ϵ is too large, we may find that $\mathbf{r}_{(x,l)}$ has steering angles that are too different from those in $sim(\mathbf{s})$. On the other hand, if ϵ is too small, we may not be able to find a $\mathbf{r}_{(x,l)}$ that is comparable to $sim(\mathbf{s})$ for many scenarios $\mathbf{s} \in S$ randomly generated in the first step. In our experiments, we select $\epsilon = 0.1$ ($2.5°$) since, based on our preliminary evaluations, we can achieve an optimal balance with this threshold.

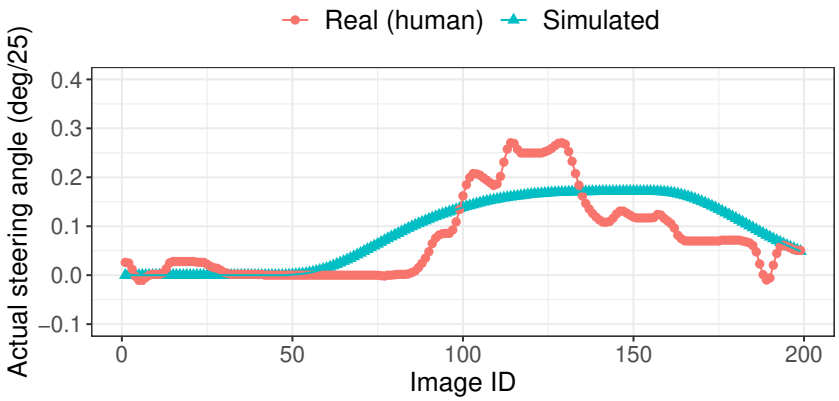
For each comparable pair of datasets $sim(\mathbf{s})$ and $\mathbf{r}_{(x,l)}$, we measure the *prediction error difference* for the same DNN to compare the datasets. Specifically, we measure $|MAE(d, sim(\mathbf{s})) - MAE(d, \mathbf{r}_{(x,l)})|$ of a DNN d. Recall that offline testing results for a given DNN d are measured based on prediction errors in terms of MAE. If $|MAE(d, sim(\mathbf{s})) - MAE(d, \mathbf{r}_{(x,l)})| \leq 0.1$ (meaning $2.5°$ of average prediction error across all images), we say that $\mathbf{r}_{(x,l)}$ and $sim(\mathbf{s})$ yield *consistent* offline testing results for d.

We note that the real-life images in the Udacity test dataset are multicolored or polychromatic. However, our preliminary evaluation confirmed that the steering predictions of our DNN subjects do not change more than $0.006°$ on average when we convert polychromatic images to monochromatic images in the Udacity test dataset. Hence, we do not attempt to make the colors of the simulator-generated images similar to that of the real-life images as color has little impact on the DNN's predictions.

Results

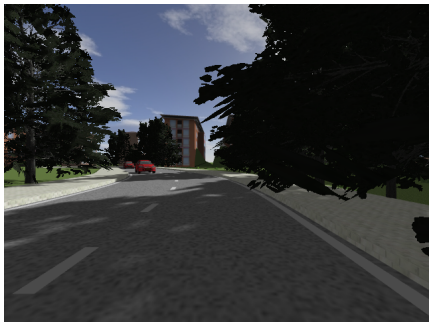
Among the 100 randomly generated scenarios (i.e., $|S| = 100$), we identified 92 scenarios that could match subsequences of the Udacity real-life test dataset. Figure 3.7 shows an example comparable pair of $\mathbf{r}_{(x,l)}$ (i.e., real dataset) and $sim(\mathbf{s})$ (i.e., simulator-generated dataset) identified using our two-step heuristic. Specifically, Figure 3.7a shows the steering angles for all the images in the example comparable pair. Figures 3.7b and 3.7c show two matching frames from the pair where the difference in the steering angles is the smallest (i.e., the 40th frames where $|\theta^r - \theta^s| = 0$). Figures 3.7d and 3.7e show two other matching frames from the pair where the difference in the steering angles is the largest (i.e., the 112th frames where $|\theta^r - \theta^s| = 0.1115$). As shown in the steering angle graph in Figure 3.7a, the simulator-generated dataset and its comparable real dataset subsequence do not have identical steering angles. For example, the actual steering angles produced by a human driver have natural fluctuations whereas the steering angles generated by the simulator are relatively smooth. The differences in steering

[3] If f has multiple points of the minima, one of them is randomly returned.



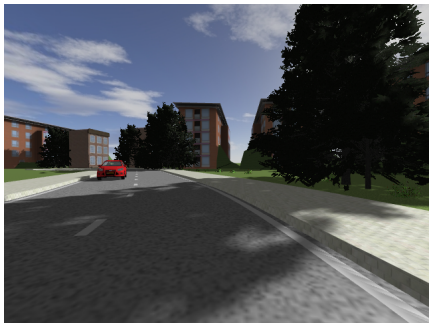
(a) Actual steering angles

(b) The 40th real image



(c) The 40th simulated image

(d) The 112th real image



(e) The 112th simulated image

Figure 3.7: Example comparable pair of a simulator-generated and real-life datasets

angles can also be attributed to the complexity of the real-world not reflected in the simulator (e.g., bumpy roads). Nevertheless, the overall steering angle patterns are very similar. If we look at the matching frames shown in Figures 3.7b and 3.7c, the matching frames look quite similar in terms of essential properties, such as road topology and incoming vehicles on the other lane. Regarding the matching frames shown in Figures 3.7d and 3.7e, they capture the largest difference in steering angles of the comparable pair of real and simulated datasets. We can note differences between the matching frames regarding some aspects, such as the shape of buildings and trees. Once again, this is because the complexity and diversity of the real-world is not fully reflected in the simulator. This point will be further discussed in Section 3.3.1.

Figure 3.8 shows, for each of our DNNs, Autumn, Chauffeur, and Komanda, the distributions of the prediction error differences for the real datasets (subsequences) and the simulator-generated datasets. For Autumn, the average prediction error difference between the real datasets and the simulator-generated

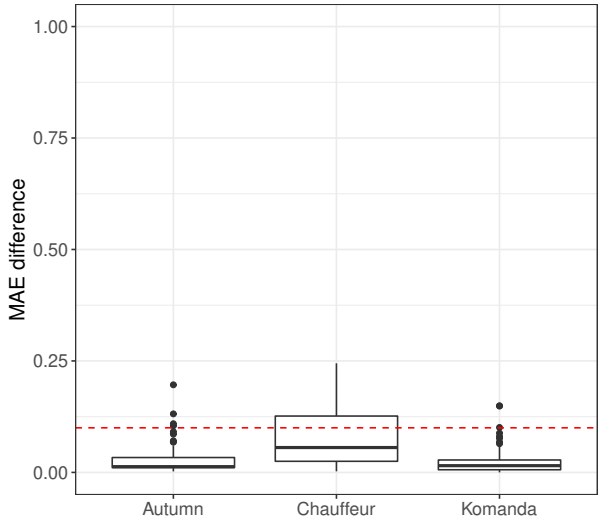


Figure 3.8: Distributions of the differences between the prediction errors obtained for the real datasets (subsequences) and the simulator-generated datasets

datasets is 0.027. Further, 95.6% of the comparable pairs show a prediction error difference below 0.1 ($2.5°$). This means that the (offline) testing results obtained for the simulator-generated datasets are consistent with those obtained using the real-world datasets for almost all comparable dataset pairs. The results for Komanda are similar: the average prediction error difference is 0.023, and 96.7% of the comparable pairs show a prediction error difference below 0.1 ($2.5°$). On the other hand, for Chauffeur, only 66.3% of the comparable pairs show a prediction error difference below 0.1. This means that testing results between real datasets and simulator-generated datasets are inconsistent in 33.71% of the 92 comparable pairs. Specifically, for *all* the inconsistent cases, we observed that the MAE value for the simulator-generated dataset is greater than its counterpart for the real-world dataset. It is therefore clear that the prediction error of Chauffeur tends to be larger for the simulator-generated dataset than for the real-world dataset. In other words, the simulator-generated datasets tend to be conservative for Chauffeur and report more false positives than for Autumn and Komanda in terms of prediction errors. We also found that, in several cases, Chauffeur's prediction errors are greater than 0.2 while Autumn's and Komanda's prediction errors are less than 0.1 for the same simulator-generated dataset. One possible explanation is that Chauffeur is over-fitted to the texture of real images, while Autumn is not thanks to the image preprocessing module. Nevertheless, the average prediction error differences between the real datasets and the simulator-generated datasets is 0.080 for Chauffeur, which is still less than 0.1. This implies that, although Chauffeur will lead to more false positives (incorrect safety violations) than Autumn and Komanda, the number of false positives is still unlikely to be overwhelming.

We remark that the choice of simulator as well as the way we generate data using our selected simulator, based on carefully designed experiments such as the ones presented here, are of great importance. Selecting a suboptimal simulator may lead to many false positives (i.e., incorrectly identified prediction errors) rendering simulator-generated datasets ineffective.

The answer to RQ1 is that, for all the subject DNNs, the prediction error differences between simulator-generated and real-life datasets are less than 0.1 on average. We conclude that we can use simulator-generated datasets as a reliable alternative to real-world datasets for testing DNNs.

3.2.3 RQ2: Comparison between Offline and Online Testing Results

Setup

We aim to compare offline and online testing results in this research question. We randomly generate scenarios and compare the offline and online testing results for each of the simulator-generated datasets.

For the scenario generation, we use the extended domain model (see Figure 3.3) to take advantage of all the feasible attributes provided by the simulator. Specifically, in Figure 3.3, the gray-colored entities and attributes in bold are additionally included in the extended domain model compared to the restricted domain model used for RQ1. For example, the (full) domain model contains various weather conditions, such as rain, snow, and fog, in addition to sunny.

Let S' be the set of randomly generated scenarios based on the (full) domain model. For each scenario $\mathbf{s} \in S'$, we prepare the simulator-generated dataset $sim(\mathbf{s})$ for offline testing and measure $MAE(d, sim(\mathbf{s}))$ for a DNN d. For online testing, we measure $MDCL(d, \mathbf{s})$. Then we compute the Spearman rank correlation coefficient ρ (rho) between $MAE(d, sim(\mathbf{s}))$ and $MDCL(d, \mathbf{s})$ to assess the overall correlation between offline and online testing results. When ρ is 0, it means that there is no monotonic relation between MAE and MDCL. The closer ρ to 1, the closer the relation between MAE and MDCL to a perfectly monotonic relation. When ρ is 1, it means that MAE systematically increases (decreases) when MDCL increases (decreases).

We further compare the offline and online testing results for individual scenarios. However, since MAE and MDCL are different metrics, we cannot directly compare them. Instead, we set threshold values for MAE and MDCL to translate these metrics into binary results (i.e., *acceptable* versus *unacceptable*) that can be compared. In particular, we interpret the online testing results of DNN d for a test scenario $\mathbf{s}$ as acceptable if $MDCL(d, \mathbf{s}) < 0.7$ and unacceptable otherwise. Note that we have $MDCL(d, \mathbf{s}) < 0.7$ when the departure from the centre of the lane observed during the simulation of $\mathbf{s}$ is less than around one meter. Based on domain expert knowledge, such a departure can be considered safe. We then compute a threshold value for MAE that is semantically similar to the 0.7 threshold for MDCL. To do so, we calculate the steering angle error that leads to the vehicle deviating from the centre of the lane by one meter. This, however, depends on the vehicle speed and the time it takes for the vehicle to reach such deviation. We assume the speed of the vehicle to be 30 km/h (i.e., the slowest vehicle speed when the vehicle is driving on normal roads) and the time required to depart from the centre of the lane to be 2.7 seconds (which is a conservative driver reaction time for braking [84]). Given these assumptions, we compute the steering angle error corresponding to a one meter departure to be around $2.5°$. Thus, we consider the offline testing results of d for $\mathbf{s}$ as acceptable if $MAE(d, sim(\mathbf{s})) < 0.1$ (meaning the average prediction error is less than $2.5°$) and unacceptable otherwise.

Table 3.2: Number of scenarios classified by offline and online testing results

(a) Autumn

	MAE < 0.1	MAE ≥ 0.1	Total
MDCL < 0.7	13	1	14
MDCL ≥ 0.7	40	36	76
Total	53	37	90

(b) Chauffeur

	MAE < 0.1	MAE ≥ 0.1	Total
MDCL < 0.7	18	0	18
MDCL ≥ 0.7	57	15	72
Total	75	15	90

(c) Komanda

	MAE < 0.1	MAE ≥ 0.1	Total
MDCL < 0.7	13	1	14
MDCL ≥ 0.7	53	23	76
Total	66	24	90

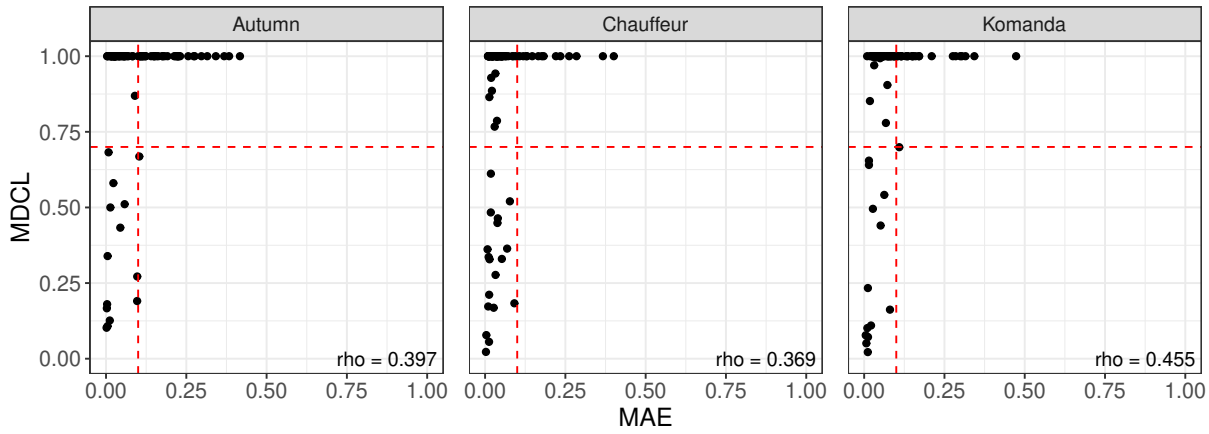
Results

Figure 3.9 shows the comparison between offline and online testing results in terms of MAE and MDCL values for all the randomly generated scenarios in S' where $|S'| = 90$. We generated 90 scenarios because it is the number of scenarios required to achieve 2-way combinatorial coverage[4] for all the attributes in our extended domain model. The x-axis is MAE (offline testing) and the y-axis is MDCL (online testing). The dashed lines represent the thresholds, i.e., 0.1 for MAE and 0.7 for MDCL. In the bottom-right corner of each diagram in Figure 3.9, we show the Spearman correlation coefficients (ρ) between MAE and MDCL. For our three DNN models, ρ is not zero but less than 0.5, meaning that there are weak correlations between MAE and MDCL.

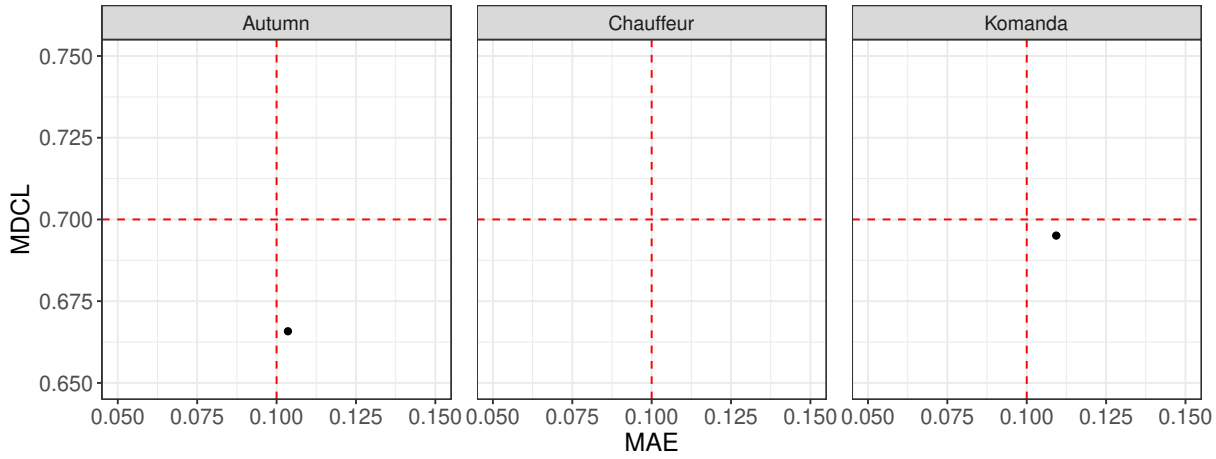
In Table 3.2, we have the number of scenarios classified by the offline and online testing results based on the thresholds. The results show that offline testing and online testing are not in agreement for 45.5%, 63.3%, and 60.0% of the 90 randomly generated scenarios for Autumn, Chauffeur, and Komanda, respectively. Surprisingly, we have only two cases (one from Autumn and one from Komanda) where the online testing result is acceptable while the offline testing result is not, and even these two exceptional cases are very close to the border line as shown in Figure 3.9b, i.e., $(0.104, 0.667)$ for Autumn and $(0.109, 0.695)$ for Komanda where (x, y) indicates MAE=x and MDCL=y. After analyzing the online testing results of these two cases in more detail, we found that MDCL was less than the threshold simply because the road was short, and would have been larger had the road been longer. Consequently, the results show that offline testing is significantly more optimistic than online testing for the disagreement scenarios.

Figure 3.10 shows one of the scenarios on which offline and online testing disagreed. As shown in Figure 3.10a, the prediction error of the DNN for each image is always less than $1°$. This means that the

[4]We use PICT (`https://github.com/microsoft/pict`) to compute combinatorial coverage.



(a) Full view

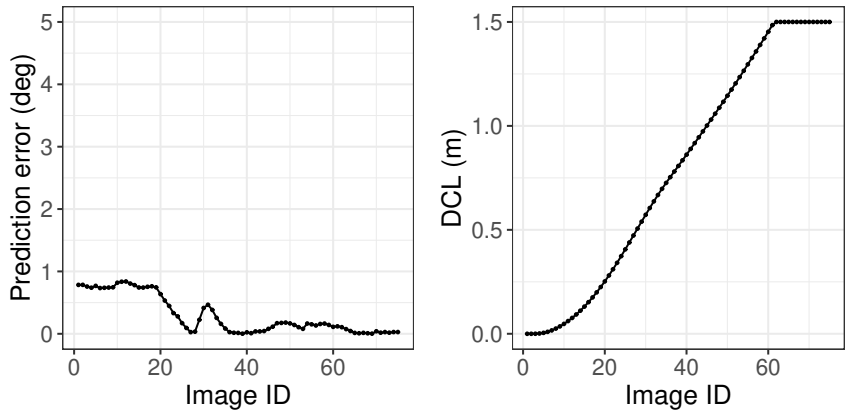


(b) Close-up of the border line

Figure 3.9: Comparison between offline and online testing results for all scenarios

DNN appears to be accurate enough according to offline testing. However, based on the online testing result in Figure 3.10b, the ego vehicle departs from the center of the lane in a critical way (i.e., more that $1.5\,\mathrm{m}$). This is because, over time, small prediction errors accumulate, eventually causing a critical lane departure. Such accumulation of errors over time is only observable in online testing, and this also explains why there is no case where the online testing result is acceptable while the offline testing result is not.

The answer to RQ2 is that offline and online testing results differ in many cases (45.5%, 63.3%, and 60.0% of all scenarios for Autumn, Chauffeur, and Komanda, respectively). We found that offline testing cannot properly reveal safety violations in ADS-DNNs, because it does not account for their closed-loop behavior. Given the fact that detecting safety violations in ADS is the ultimate goal of ADS-DNN testing, we conclude that online testing is preferable to offline testing for ADS-DNNs.



(a) Offline testing result (b) Online testing result

Figure 3.10: Example inconsistent results between offline and online testing

3.2.4 RQ3: Rule Extraction

Setup

In RQ2, we showed that, for ADS-DNNs, offline prediction errors are not correlated with unsafe deviations observed during online testing. In other words, offline testing will not reveal some of the safety violations that can be revealed via online testing. However, offline and online testing are both essential steps in development and verification of DNNs [147]. A typical workflow for DNN testing is to first apply offline testing, which is a standard Machine Learning process, and then move to online testing, which is more expensive and requires engineers to invest significant time on integrating the DNN into a simulated application environment. The goal of this research question is to provide guidelines on how to combine offline and online testing results to increase the effectiveness of our overall testing approach (i.e., to reveal the most faults) while reducing testing cost. To achieve this goal, we identify conditions specified in terms of our domain model attributes (Figure 3.3) that characterize when offline and online testing agree and when they disagree. We seek to derive these conditions for different DNN subjects. Provided with these conditions, we can identify testing scenarios that should be the focus of online testing, i.e., scenarios for which offline testing is ineffective, but online testing may reveal a safety violation.

In our work, as discussed in Section 3.1.3, each test scenario is specified as a vector of values assigned to the attributes in our domain model. By applying offline and online testing to all test scenario vectors, we can determine if it belongs to the category where offline and online testing results are in agreement or not. Using test vectors and their corresponding categories as a set of labelled data instances, we can then generate classification rules by applying well-known rule mining algorithms, such as RIPPER [23], to learn conditions on domain model attributes that lead to agreement or disagreement of offline and online testing results. To be able to learn these conditions with a high degree of accuracy, however, we need to gather a large collection of labelled data instances including test vectors ideally covering all combinations of value assignments to the attributes of our domain model. However, the number of all combinations of all the attribute values is more than 2^{32} since we have 32 attributes of enumeration types in our domain model that can take more than two values. Furthermore, the simulation time on a desktop with a 3.6 GHz Intel i9-9900k processor with 32 GB memory and graphic card Nvidia GeForce RTX 2080Ti is up to 20-30 minutes for each scenario depending upon attributes like road length and speed of

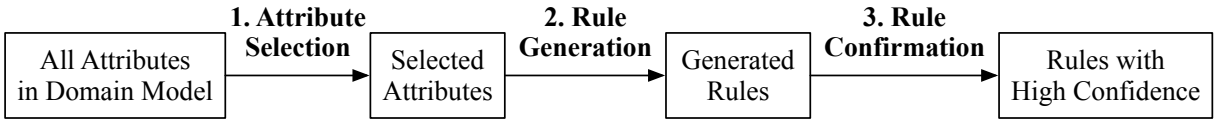


Figure 3.11: Overall workflow of the three-step heuristic approach to extract rules.

ego vehicle. Therefore, it is impossible to cover all the combinations of value assignments to our domain model attributes.

To be able to learn conditions characterizing agreement and disagreement between offline and online testing in an effective and efficient way, we propose a three-step heuristic approach that focuses on learning rules with statistically high confidence while minimizing the number of test vectors (i.e., value assignments to domain model attributes) required for learning. Specifically, we incrementally generate new test vectors to be labelled by focusing on specific attributes to minimize the amount of data needed for classification while increasing statistical significance.

Figure 3.11 outlines the workflow of the approach. The first step is *attribute selection*, which aims to reduce the search space by identifying a subset of attributes correlated with the differences between offline and online testing results for a DNN. The second step (*rule generation*) aims at extracting rules, for a given DNN, based on the selected attributes. The third step, *rule confirmation*, seeks to improve the statistical confidence of the extracted rules. In each of the steps, a minimal number of new data instances are incrementally generated. The details of the steps are described next:

1. *Attribute Selection*: In data mining, attribute selection (a.k.a., feature selection) strategies often rely on generating a large number of labelled data instances randomly to ensure data diversity and the uniformity of their distribution in the search space. Since labelling data instances (i.e., test vectors) in our work is expensive, we are limited regarding how many test vectors we can generate and label for the purpose of attribute selection. Therefore, instead of using a pure random strategy, as the input for our attribute selection strategy, we use n-way combinatorial testing to generate a relatively small number of diverse test vectors. The value of n is determined based on our time budget for labelling data (running test vectors in our work) and the number of attributes in our domain. In our work, for the purpose of attribute selection, we set $n = 2$. This led to generating 90 test vectors to be able to cover the pairwise combination of values of the 32 attributes in our domain model. For each test vector, we determine whether offline and online testing results are in agreement or not, resulting in the binary classification of our test vectors. We then perform the attribute selection using the Random Forest algorithm [44]. We use the concept of variable importance in Random Forests to select important attributes in our domain model since it has been reported to be accurate in general [7, 44]. Table 3.3 shows the selected attributes for each DNN. Six, four, and two attributes are selected for Autumn, Chauffeur, and Komanda, respectively.

2. *Rule Generation*: At this step, we trim the set of test vectors generated in the attribute selection step by hiding, from each vector, the attributes that were not selected in the previous step. The result is a set of labelled test vectors that only include values for the important attributes of our domain (i.e., those attributes selected in the previous step). This set, however, does not necessarily cover different combinations of the values for the selected attributes. Hence, we enhance this set by generating a number of new test vectors. We use n-way combinatorial test generation again, but

Table 3.3: Intermediate Results: Selected Attributes

DNN	Selected Attributes
Autumn	*Road.type*, *Road.laneLineColor*, *Road.curbLinePattern*, *Vehicle.laneNumber*, *Vehicle.headLights*, *Weather.condition*
Chauffeur	*Road.type*, *Road.roadSpecificProperty*, *Vehicle.fogLights*, *Environment.underlay*
Komanda	*Weather.type*, *Environment.bulidings*

this time we only consider the attributes selected in the previous step in the test generation (i.e., the attributes that were not selected in step one are simply set a random value). For this step, we choose $n = 3$ since we are dealing with a smaller number attributes and are able to generate more value combinations during the same test time budget. For Komanda, we use all combinations since we have only two selected attributes. The number of new test vectors therefore varies depending on the selected attributes for each DNN. The new test vectors, together with the 90 test vectors generated in the previous step, are used for rule generation. We extract rules using the RIPPER algorithm [23], yielding a set of rules for our DNN under analysis. Each generated rule is a tuple $(if, label)$ where if describes conditions on the values of the selected attributes (e.g., *Vehicle.speed* $> 10 \wedge$ *Road.type* = *Curved*) and $label$ describes a class (i.e., *agree* or *disagree*). We can estimate the accuracy of each rule as the number of test vectors labelled by $label$ and satisfying the if conditions over the number of test vectors satisfying if. Table 3.4 shows the rules generated for each DNN. As shown in the table, we obtain three rules for Autumn, four rules for Chauffeur, and two rules for Komanda. Each rule has an if part, described as a conjunction of predicates defined over our domain model attributes, and a $label$ part that can be either *agree* or *disagree*. For the accuracy values, we also report the 95% Confidence Interval (CI). For example, for the second rule of Autumn, the accuracy value 0.88 ± 0.20 means that the true accuracy value has a 95% probability of being in the interval [0.68 1.00]. The CI range is quite large since, in our work, we have minimized the total number of test vectors, and therefore the number of test vectors satisfying the conditions if for each rule can be small. Hence we may not be able establish reasonably narrow CIs for the accuracy values of the generated rules. To alleviate this issue, we use a third step to increase the statistical confidence in the estimated accuracy of the generated rules.

3. *Rule Confirmation*: The basic idea is to reduce the CI length by providing more data instances for each rule. For each rule, we repeatedly generate a data instance satisfying the if part of the rule until the CI length of the estimated accuracy of the rule with a 95% confidence level is less than a threshold λ. In our experiments, we set $\lambda = 0.2$, meaning ± 0.1. Note that we generate new data instances only for the extracted rules to efficiently reduce the CI length of the rules. The final results after performing the Rule Confirmation step is shown in Table 3.5 and will be discussed in the next section (Section 3.2.4).

Results

Table 3.5 shows the rules generated for our three DNN subjects after applying the process described in Section 3.2.4 and Figure 3.11. Note that in contrast to the results reported in Table 3.4, the accuracy

Table 3.4: Intermediate Results: Generated Rules

DNN	Rule	Accuracy
Autumn	If *Road.curbLanePattern* = *Dashed* then *disagree*	0.58 ± 0.11
	If *Road.type* = *Curved* then *disagree*	0.88 ± 0.20
	Other than mentioned above then *agree*	0.65 ± 0.11
Chauffeur	If *Vehicle.fogLights* = *True*, then *agree*	0.59 ± 0.13
	If *Road.type* = *Straight* $\wedge$ *Environment.underlay* = *Pavement* then *agree*	0.90 ± 0.18
	If *Road.type* = *Curved* then *agree*	0.70 ± 0.28
	Other than mentioned above then *disagree*	0.76 ± 0.09
Komanda	If *Weather.type* = *Rainy* $\wedge$ *Environment.buildings* = *False* then *agree*	0.76 ± 0.18
	Other than mentioned above then *disagree*	0.69 ± 0.11

Table 3.5: Rule Extraction Results

DNN	ID	Rule	Accuracy
Autumn	A1	If *Road.curbLanePattern* = *Dashed* then *disagree*	0.61 ± 0.10
	A2	If *Road.type* = *Curved* then *disagree*	0.95 ± 0.10
	A3	Other than mentioned above then *agree*	0.61 ± 0.10
Chauffeur	C1	If *Vehicle.fogLights* = *True*, then *agree*	0.58 ± 0.10
	C2	If *Road.type* = *Straight* $\wedge$ *Environment.underlay* = *Pavement* then *agree*	0.71 ± 0.10
	C3	If *Road.type* = *Curved* then *agree*	0.55 ± 0.10
	C4	Other than mentioned above then *disagree*	0.76 ± 0.09
Komanda	K1	If *Weather.type* = *Rainy* $\wedge$ *Environment.buildings* = *False* then *agree*	0.59 ± 0.10
	K2	Other than mentioned above then *disagree*	0.67 ± 0.10

values in Table 3.5 are those obtained after applying the rule confirmation step. For example, for the first rule for Autumn, the accuracy of 0.61 ± 0.10 means that around 61% of the scenarios that satisfy the condition *Road.curbLanePattern* = *Dashed* are labelled with *disagree*. Thanks to our rule confirmation step, we are able to ascertain the accuracy levels of rules within a narrower 95% confidence interval. In our work, the rule accuracy indicates the predictive power of the rule. For example, the second rule for Autumn is highly accurate and hence predictive (more than 85% of scenarios). Hereafter, for simplicity, we use the IDs indicated in Table 3.5 to refer to the rules.

Overall, there is no rule predicting *agree* with an accuracy above 0.71. This means that there is no condition with accuracy above 0.71 where offline testing results conform to online testing results. That is, the test results for scenarios that match the "agree" rule conditions in Table 3.5 may still differ during offline and online testing with a high probability. Therefore, based on our results, we are not able to identify conditions that can characterize, with a high accuracy, agreement between offline and online testing to help lift offline testing results to online testing and reduce the amount of online testing needed. Our results, further, suggest that, at least for ADS-DNNs, we may not be able to find rules that can, in general, differentiate between offline and online testing behaviors. As can be seen from the table, there is not much similarity between the rules we have obtained for different DNNs. This is because these DNNs have different architectures, use different features of the input images for prediction and are trained differently.

However, our observations show that these rules still may provide valuable insights as to how different DNNs work. When an attribute appears in a rule, it indicates that the attribute has a significant impact on the DNN output, and hence, this attribute can be used to classify both the situations where offline testing is as good as online testing (i.e., DNN prediction errors indeed indicate a safety violation) as well as the dual situations where offline testing is simply too optimistic. For example, the attributes *Weather.type* and *Environment.buildings* appear only in the conditions for the rules of Komanda. On the other hand, for Chauffeur and Autumn, the attributes appearing in the rule conditions are related to the road shape and the road lane patterns. This confirms the fact that Komanda uses full images to predict steering angles while the Chauffeur and Autumn focus on the road-side views in the images, as noted in Section 3.2.1.

Another reason explaining differences across DNNs is that some DNNs are inaccurate for certain attributes regardless of testing modes, which means that both offline testing and online testing are capable of detecting the faulty behaviors of these DNNs with a relatively high probability. For example, we found that Chauffeur is, in general, inaccurate for predicting steering angles for curved roads. Due to this weakness, both offline and online testing results are in agreement when *Road.type* is *Curved*, as shown in C3.

The last reason for differences is that, as shown in RQ1, Chauffeur works relatively better on real-world images than on simulated images. Since Chauffeur is not effective with simulated images, it may yield more prediction errors in offline testing, and hence, offline and online testing results are more likely to be in agreement as "unacceptable". This explains why we have three "agree" rules, namely C1, C2 and C3, for Chauffeur while for other DNNs we have fewer "agree" rules.

The answer to RQ3 is, based on our results, that offline testing results cannot be used to reduce the cost of online testing as we are not able to identify conditions that characterize, with a high accuracy, agreement between offline and online testing for all our subject DNNs.

3.2.5 Threats to Validity

In RQ1, we propose a two-step approach that builds simulator-generated datasets comparable to a given real-life dataset. While it achieves its objective, as shown in Section 3.2.2, the simulated images are still different from the real images. However, we confirmed that the prediction errors obtained by applying our subject DNNs to the simulator-generated datasets are comparable with those obtained for their corresponding real-life datasets. Thus, the conclusion that offline and online testing results often disagree with each other is valid.

We used a few thresholds that may affect the experimental results in RQ2 and RQ3. To reduce the chances of misinterpreting the results, we selected intuitive and physically interpretable metrics to evaluate both offline and online test results (i.e, prediction errors and safety violations), and defined threshold values based on common sense and experience. Further, adopting different threshold values, as long as they are within a reasonable range, does not change our findings. For example, if we use $MAE(d, sim(\mathbf{s})) < 0.05$ as a threshold in offline testing results instead of $MAE(d, sim(\mathbf{s})) < 0.1$, the numbers of scenarios in Table 3.2 change. However, it does not change the correlation analysis results and the fact that we have many scenarios for which offline and online testing results disagree, nor does it change the conclusion that offline testing is more optimistic than online testing.

Different ADS-DNNs may lead to different results. For example, we may able to identify conditions that can characterize, with a high accuracy, agreement between offline and online testing results to lift offline testing results to online testing and reduce the amount of online testing needed for a specific ADS-DNN. To mitigate such a threat, we tried our best to find all candidate ADS-DNNs in the literature and selected the three subject ADS-DNNs (i.e., Autumn, Chauffeur, and Komanda) that are publicly available and sufficiently accurate for steering angle predictions.

Though we focused, in our case study, on only two lane-keeping DNNs (steering prediction)—which have rather simple structures and do not support braking or acceleration, our findings are applicable to all DNNs in an ADS context as long as the closed-loop behavior of the ADS matters.

3.3 Discussion

3.3.1 Online Testing using Simulators

One important purpose of online testing is to test a trained ADS-DNN with the newest unseen data that potentially appear in the application environments of the ADS-DNN. However, because simulators cannot express all the complexity and diversity of the real world, online testing using simulators cannot cover all possible scenarios in the real world. For example, in the case of online testing using a simulator that cannot express weather changes, certain safety violations of the ADS-DNN that occurs only in rainy weather cannot be found.

Nevertheless, considering the problems of online testing in the real world, especially the cost and risk, online testing using a simulator is inevitable. Indeed, according to our industrial partners in the automotive industry, due to the excessive amount of manpower and resources required to collect real-world data, it is impossible to gather sufficient and diverse real-world data. On the other hand, a simulator can generate sufficiently diverse data at a much lower cost and risk.

Furthermore, simulator-based test input generation has an additional advantage regarding the test oracle problem [10]. When we use simulators for online testing, the generation of test oracles is completely automated. For real-life datasets, however, test oracles may need to be manually specified which is labor-intensive and time-consuming. For example, for ADS-DNNs, the driver's maneuvers and the data gathered from the various sensors and cameras during online testing in the real world may not contain sufficient information to automatically generate test oracles. In contrast, simulators are able to generate labeled datasets, from which test oracles can be automated, for various controlling, sensing, and image recognition applications. But, as expected, the accuracy of test results and oracles depends on the fidelity of simulators.

3.3.2 Offline vs. Online Testing: What to Use in Practice?

Experimental results show that offline testing cannot properly detect safety requirements violations identified in online testing. Offline testing is in fact inadequate to identify faulty behaviors for ADS-DNNs with closed-loop behavior. In other words, online testing is essential to adequately detect safety violations in ADS-DNNs, where interactions with the application environment are important. In particular, online testing using a simulator is highly recommended if a high-fidelity simulator is available.

However, online testing is not essential in all cases. When testing ADS-DNNs without closed-loop behavior, offline and online testing results are expected to be similar because errors are not accumulating

over time. For example, in the case of an ADS-DNN that simply warns the driver instead of directly controlling the steering when necessary, there is no closed-loop since the DNN's predictions do not actually control the vehicle, and therefore offline testing would be sufficient.

3.3.3 Open Challenges

There are also challenges that need to be addressed in online testing. For example, the higher the fidelity of a simulator, the more time it takes to simulate, which has a direct impact on the cost of online testing. In particular, when using a search-based technique, online testing may take a very long time because the simulator must be repeatedly executed for various scenarios. Therefore, more research is required to reduce the cost of online testing.

Research on high-fidelity simulators is also essential. As discussed in Section 3.3.1, online testing using a simulator cannot completely cover all possible scenarios in the real world. However, by utilizing a simulator higher fidelity, the risk of uncovered scenarios could be significantly reduced [112, 106]. Research on how to lower the risk through a more systematic approach is also needed.

3.4 Related Work

Table 3.6 summarizes DNN testing approaches specifically proposed in the context of autonomous driving systems. Approaches to the general problem of testing machine learning systems are discussed in the recent survey by [147].

In Table 3.6, approaches for online testing are highlighted grey. As the table shows, most of existing approaches focus on the offline testing mode only, where DNNs are seen as individual units without accounting for the closed-loop behavior of a DNN-based ADS. Their goal is to generate test data (either images or 3-dimensional point clouds) that lead to DNN prediction errors. Dreossi et al. [32] synthesized images for driving scenes by arranging basic objects (e.g., road backgrounds and vehicles) and tuning image parameters (e.g., brightness, contrast, and saturation). Pei et al. [98] proposed DEEPXPLORE, an approach that synthesizes images by solving a joint optimization problem that maximizes both neuron coverage (i.e., the rate of activated neurons) and differential behaviors of multiple DNNs for the synthesized images. Tian et al. [125] presented DEEPTEST, an approach that generates label-preserving images from training data using greedy search for combining simple image transformations (e.g., rotate, scale, and for and rain effects) to increase neuron coverage. Wicker et al. [139] generated adversarial examples, i.e., small perturbations that are almost imperceptible by humans but causing DNN misclassifications, using feature extraction from images. Zhang et al. [148] presented DEEPROAD, an approach that produces various driving scenes and weather conditions by applying Generative Adversarial Networks (GANs) along with corresponding real-world weather scenes. Zhou and Sun [155] combined Metamorphic Testing (MT) and Fuzzing for 3-dimensional point cloud data generated by a LiDAR sensor to reveal erroneous behaviors of an object detection DNN. Zhou et al. [153] proposed DEEPBILLBOARD, an approach that produces both digital and physical adversarial billboard images to continuously mislead the DNN across dashboard camera frames. While this work is different from the other offline testing studies as it introduces adversarial attacks through sequences of frames, its goal is still the generation of test images to reveal DNN prediction errors. In contrast, Kim et al. [64] defined a coverage criterion, called *surprise adequacy*, based on the behavior of DNN-based systems with respect to their training data. Images generated by

Table 3.6: Summary of DNN testing studies in the context of autonomous driving

Author(s)	Testing mode	DNN's role	Summary
Dreossi et al. [32]	Offline	Object detection	Test image generation by arranging basic objects using greedy search
Pei et al. [98]	Offline	Lane keeping	Coverage-based label-preserving test image generation using joint optimization with gradient ascent
Codevilla et al. [22]	Offline and online	Lane keeping	Improving the correlation between offline and online testing results by selecting an appropriate testing dataset and suitable offline metrics
Tian et al. [125]	Offline	Lane keeping	Coverage-based label-preserving test image generation using greedy search with simple image transformations
Tuncali et al. [126]	Online	Object detection	Test scenario generation using the combination of covering arrays and simulated annealing
Wicker et al. [139]	Offline	Traffic sign recognition	Adversarial image generation using feature extraction
Zhang et al. [148]	Offline	Lane keeping	Label-preserving test image generation using Generative Adversarial Networks (GANs)
Zhou et al. [153]	Offline	Lane keeping	Adversarial billboard-image generation for digital and physical adversarial perturbation
Gambi et al. [40]	Online	Lane keeping	Automatic virtual road network generation using search-based Procedural Content Generation (PCG)
Kim et al. [64]	Offline	Lane keeping	Improving the accuracy of DNNs against adversarial examples using surprise adequacy
Majumdar et al. [82]	Online	Object detection, lane keeping	Test scenario description language and simulation-based test scenario generation to cover parameterized environments
Zhou and Sun [155]	Offline	Object detection	Combination of Metamorphic Testing (MT) and fuzzing for 3-dimensional point cloud data
This chapter	Offline and online	Lane keeping	Comparison between offline and online testing results and investigate if we can use offline testing results to run fewer tests during online testing

DEEPTEST were sampled to improve such coverage and used to increase the accuracy of the DNN against adversarial examples.

Online testing studies exercise the ADS closed-loop behavior and generate test driving scenarios that cause safety violations, such as unintended lane departure or collision with pedestrians. Tuncali et al. [126] were the first to raise the problem that previous works mostly focused on the DNNs, without accounting for the closed-loop behavior of the system. Gambi et al. [40] also pointed out that testing DNNs for ADS using only single frames cannot be used to evaluate closed-loop properties of ADS. They presented ASFAULT, a tool that generates virtual roads which cause self-driving cars to depart from their lane. Majumdar et al. [82] presented a language for describing test driving scenarios in a parametric way and provided PARACOSM, a simulation-based testing tool that generates a set of test parameters in such a way as to achieve diversity. We should note that all the online testing studies rely on virtual (simulated) environments, since, as mentioned before, testing DNNs for ADS in real traffic is dangerous and expensive. Further, there is a growing body of evidence indicating that simulation-based testing is effective at finding violations. For example, recent studies for robotic applications show that simulation-based testing of robot function models not only reveals most bugs identified during outdoor robot testing, but that it can additionally reveal several bugs that could not have been detected by outdoor testing [117].

There is only one study comparing offline and online testing results by investigating the correlations between offline and online testing prediction error metrics [22]. The authors found that the correlation between offline prediction and online performance is weak, which is consistent with the results of this chapter. They also found two ways for improving the correlations: (1) augmenting the testing data (e.g., include images from three cameras, i.e., a forward-facing one and two lateral cameras facing 30 degrees left and right, instead of having images from one forward-facing camera) and (2) selecting a proper offline testing metric (e.g., Mean Absolute Error other than Mean Squared Error). Their analysis relies on the offline and online testing of DNNs trained by simulator-generated images, while our DNNs are trained with real-world images. Nevertheless, consistent with our results, they concluded that offline testing is not adequate. Furthermore, beyond simple correlations and in order to draw more actionable conclusions, our investigation looked at whether offline testing was a sufficiently reliable mechanism for detecting safety violations in comparison to online testing. Last, we investigated whether offline and online testing results could agree under certain conditions, so as to take advantage of the lower cost of offline testing in such situations.

3.5 Conclusion

This chapter presents a comprehensive case study to compare two distinct testing phases of Deep Neural Networks (DNNs), namely offline testing and online testing, in the context of Automated Driving Systems (ADS). Offline testing evaluates DNN prediction errors based on test data that are generated without involving the DNN under test. In contrast, online testing determines safety requirement violations of a DNN-based system in a specific application environment based on test data generated dynamically from interactions between the DNN under test and its environment. We aimed to determine *how offline and online testing results differ or complement each other* and *if we can exploit offline testing results to run fewer tests during online testing to reduce the testing cost*. We additionally investigated if we can use simulator-generated datasets as a reliable substitute to real-world datasets for DNN testing.

The experimental results on the three best performing ADS-DNNs from the Udacity Self-Driving Car Challenge 2 [128] show that simulator-generated datasets yield DNN prediction errors that are similar to those obtained by testing DNNs with real-world datasets. Also, offline testing is more optimistic than online testing as many safety violations identified by online testing could not be identified by offline testing, while large prediction errors generated by offline testing always led to severe safety violations detectable by online testing. Furthermore, the experimental results show that we cannot exploit offline testing results to reduce the cost of online testing in practice since we are not able to identify specific situations where offline testing could be as accurate as online testing in identifying safety violations.

The results of this chapter have important practical implications for DNN testing, not only in an ADS context but also in other CPS where the closed-loop behavior of DNNs matters. Specifically, both researchers and practitioners should focus more on online testing as offline testing is not able to properly determine safety requirement violations of the DNN-based systems under test.

Chapter 4

Automatic Test Suite Generation for Key-Points Detection DNNs using Many-Objective Search

In this chapter, we present an approach to automatically generate test data for Key-Points detection DNNs (KP-DNNs) using many-objective search. Automatically detecting key-points (e.g., facial key-points or finger key-points) in an image or a video is a fundamental step for many applications, such as face recognition [149], facial expression recognition [74], and drowsiness detection [59]. With the recent advances in Deep Neural Networks (DNNs), Key-Points detection DNNs (KP-DNNs) have been widely studied [57, 58, 59, 116].

To ensure the reliability of KP-DNNs, it is essential to check how accurate the DNNs are when applied to various test data. Nevertheless, testing KP-DNNs is still often focused on pre-recorded test data, such as publicly available datasets [143, 6, 110] that are typically not collected or generated according to any systematic strategy, and therefore provide limited test results and confidence. Further, since 3rd parties with ML expertise often provide such KP-DNNs, the internal information of the KP-DNNs is usually not available to the engineers who integrate them into their application systems [77, 97]. As a result, black-box testing approaches should be favored, preferably one that is dedicated and adapted to KP-DNNs to maximize test effectiveness within acceptable time constraints, which is the objective of this chapter.

To automatically generate new and diverse test data, one may opt to use advanced DNN testing approaches that have been recently proposed by several researchers. For example, Tian et al. [125] presented DeepTest, an approach that generates new test images from training images by applying simple image transformations (e.g., rotate, scale, and for and rain effects). Wicker et al. [140] generated adversarial examples, i.e., small perturbations that are almost imperceptible by humans but causing DNN misclassifications, using feature extraction from images. Gambi et al. [41] presented AsFault, an approach that generates virtual road networks in computer simulations using search-based testing for testing a DNN-based automated driving system. However, none of the existing DNN testing approaches take into

account the following testing challenges that are specific to KP-DNNs predicting multiple key-points at the same time. First, accuracy for individual key-points is of great importance, and therefore one should not simply consider average prediction errors across all key-points. Second, the number of key-points is typically large, e.g., our evaluation uses a facial KP-DNN that detects 27 key-points in an image. Third, depending on the performance of the KP-DNN under test, it may be infeasible to generate a test image causing a significant misprediction for some particular key-points. Therefore, if such infeasibility is observed at run time, it is essential to dynamically and efficiently distribute the computational resources dedicated to testing to the other key-points. This implies that using one of the existing DNN testing approaches is not an option. It also prevents us from running an independent search for each individual key-point.

To address the above challenges, we recast the problem of KP-DNN testing as a many-objective optimization problem. Since the severe misprediction of each individual key-point becomes one objective, many-objective optimization algorithms can potentially be effective and efficient at generating test data that causes the KP-DNN to mispredict individual key-points. Though our approach is applicable to any KP-DNNs, we focus our experiments on a particular but common industrial application, Facial KP-DNNs (FKP-DNN). The experimental subjects are provided by our industry partner IEE in the automotive domain, who is working on driver gaze detection systems that either warn the driver or take the control of the vehicle when the driver is apparently not paying attention on the road while driving. Our empirical investigation shows that our approach is effective at generating test data that cause most of the key-points (93% on average) to be severely mispredicted. Though many of these mispredictions are not avoidable because a key-point can be invisible due to, for example, a shadow, characterizing them is important to analyze risks and a large number of them can still be addressed by retraining or other means. We further investigate (1) the degree of mispredictions caused by test data generated by alternative search algorithms and (2) how to learn specific conditions (e.g., head posture) leading to severe mispredictions of individual key-points. Our ultimate goal is to provide practical recommendations on how to test KP-DNNs and how to analyze test results to support risk analysis and retraining.

The contributions of this chapter are summarized as follows:

- We formalize the problem definition of KP-DNN testing. Our formalism specifies KP-DNN input and output variables, as well as the notion of severe misprediction for a key-point, and characterises the specific testing challenges in this context.
- We propose a way to automatically generate test data for KP-DNNs using many-objective search algorithms and simulation.
- We investigate how automatically generated test data and results can be effectively used to reveal and possibly address the root causes of inaccuracy in FKP-DNNs under test.
- We present empirical results and lessons learned drawn from our experience in applying the approach in an industrial context.

The rest of the chapter is organized as follows. Section 4.1 formalises the problem of KP-DNN testing. Section 4.2 describes our approach. Section 4.3 evaluates our approach with an industrial case study and summarizes lessons learned. Section 4.4 positions our work with respect to related work. Section 4.5 concludes the chapter.

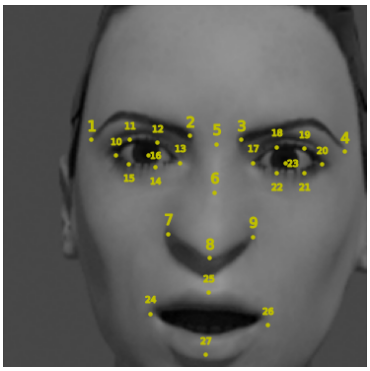


Figure 4.1: The actual positions of 27 facial key-points

4.1 Problem Definition

In this section, we provide a general but precise problem description regarding test data generation for FKP-DNNs. Though we use a specific application domain for the purpose of exemplification, this description can easily be generalized to all situations where key-points must be detected in an image, such as hand key-point detection [115] and human pose key-points detection [93], regardless of the content of that image.

A FKP-DNN takes as input a facial image (real or simulated) and returns the predicted positions of its key-points; Figure 4.1 shows an example image with the actual positions of key-points. An image can be defined by various factors, such as face size, skin color, and head posture. Using a labeled image that contains the actual positions of key-points, one can measure the degree of prediction errors for an FKP-DNN under test. The goal of test suite generation for FKP-DNNs is to generate labeled images that cause an FKP-DNN under test to inaccurately predict the positions of key-points, to an extent which affects the safety of the system relying on such predictions.

More specifically, let t be a labeled test image composed of a tuple $t = (\mathbf{ic}_t, \mathbf{p}_t)$ where $\mathbf{ic}_t$ represents image characteristics, such as face size, skin color, and head posture, and $\mathbf{p}_t$ represents the actual positions of key-points in t. $\mathbf{p}_t$ can be further decomposed as $\mathbf{p}_t = \langle p_{t,1}, \ldots, p_{t,k} \rangle$ where $p_{t,i}$ for $i = 1, \ldots, k$ is the actual position of the i-th key-point in t and k is the total number of key-points. Depending on $\mathbf{ic}_t$, some key-points may not actually be visible in t. For example, if the head is turned 90° to the right, the right eye key-point will be invisible in the image taken from the front camera. In this case, the actual positions of invisible key-points are *null*. $V(t)$ is a set of indices for visible key-points in t, and $|V(t)|$ is the total number of visible key-points in t. An FKP-DNN d can be considered as a function $d(t) = \langle \hat{p}_{t,1}, \ldots, \hat{p}_{t,k} \rangle$ where $\hat{p}_{t,i}$ is the predicted position of the i-th key-point in t. The Normalized Mean Error (NME) of d for all key-points in t can be defined as follows:

$$NME(d,t) = \frac{\Sigma_{i \in V(t)} NE(p_{t,i}, \hat{p}_{t,i})}{|V(t)|}$$

where $NE(p_{t,i}, \hat{p}_{t,i})$ is the Normalized Error of the predicted position $\hat{p}_{t,i}$ with respect to the actual position $p_{t,i}$, ranging between 0 and 1. For example, in our case study, if one measures the error using the Euclidean distance between the actual and predicted positions in an image, then the normalization can be done by dividing the distance by the height or width of the face[1], whichever is larger. Such a normalization allows error values to be compared for faces of different sizes.

[1]This can be easily calculated with the actual positions of key-points.

Generating a critical test image t for d such that it maximizes $NME(d, t)$ could be the goal of test case generation. However, it does not properly account for the fact that accuracy for individual key-points is of great importance, and therefore that averaging prediction errors across key-points may be misleading. This is, indeed, a major concern for our industry partner IEE, who is developing DNN-based gaze detection systems that either warn the driver or take the control of the vehicle when the driver is not paying attention while driving. By considering individual key-points, we may observe that certain key-points' predictions are particularly inaccurate. Such information can be used by IEE to (1) focus the retraining of d on these key-points for improving its accuracy or (2) select accurate key-points to be primarily relied upon—when there are alternative choices—by the applications using them to make decisions.

To verify if d correctly predicts individual key-points for t, we need to check if $NE(p_{t,i}, \hat{p}_{t,i})$ is less than a small threshold ϵ for all key-points. Therefore, though that may not be possible, test suite generation aims to generate a minimal set of test cases TS that satisfies $NE(p_{t,i}, \hat{p}_{t,i}) \geq \epsilon$ for all $i = 1, \ldots, k$ for some $t \in TS$. Note that the value of ϵ is usually application specific; for example, IEE uses $\epsilon = 0.05$ because $NME > 0.05$ is considered critical in their applications.

Test suite generation for FKP-DNNs entails several challenges. First, the test input space is too large to be exhaustively explored. For IEE, this is clear when considering the features that characterize a facial image, including head posture and facial characteristics, such as different shapes of eyes, noses, and mouths. Second, the number of key-points is typically large (e.g., 27 key-points for IEE's DNN) and, increasing the complexity of the problem, individual key-points are independent as people may exhibit different relative positions for the same key-points. Third, depending on the accuracy of the DNN under test, it may be infeasible to find an image causing the normalized error to exceed the threshold for some key-points. Considering a limited time test budget, if finding a critical image for a key-point seems infeasible, it is essential to dynamically and efficiently distribute computation time at run time to find critical images for the other key-points. Fourth, it is not easy to manually label the actual positions of key-points for a large number of test images. While there are publicly available real-world datasets for various key-point detection problems [143, 6, 110], such datasets limit the exploration of the input space to what data is available since there are no publicly available simulators to generate additional images. Last but not least, FKP-DNNs, like other DNNs in practice, are often provided by third parties with expertise in ML, who typically do not provide access to the trained DNN's internal information. Therefore, test suite generation for FKP-DNNs should be black-box, in the sense that it should not rely on DNN internal information.

To address the above challenges, as described in detail in Section 4.2, we suggest to apply many-objective search algorithms, which can be effective and efficient for achieving many independent objectives within a limited time budget. These algorithms are a priori a good match to our problem since requirements for individual key-points (i.e., $NE(p_{t,i}, \hat{p}_{t,i}) \geq \epsilon$ for $t \in TS$) can be translated into objective functions. Furthermore, this approach is DNN-agnostic as it considers the DNN under test as a black-box.

4.2 Search-based Test Suite Generation

This section provides a solution to the problem of test suite generation for KP-DNNs, described in Section 4.1, by applying many-objective search algorithms. Similar to Section 4.1, although our approach is not specific to the domain of facial key-points, to root the presentation into concrete examples, we describe the approach in the context of test suite generation for FKP-DNNs.

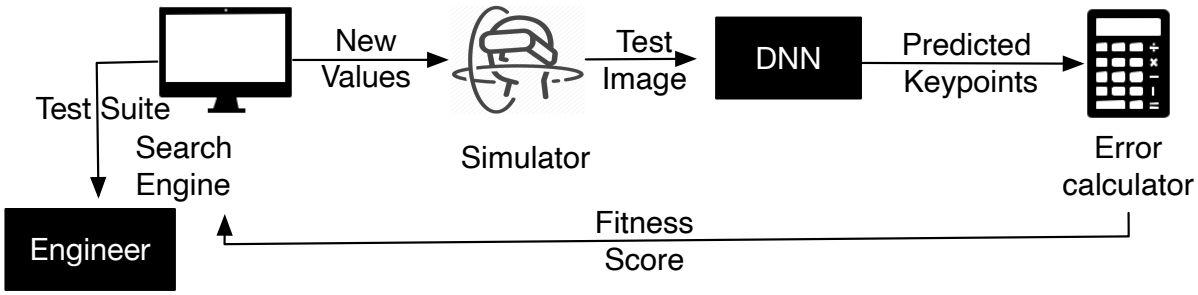


Figure 4.2: Overview of automatic test suite generation for FKP-DNNs

Recall that MOSA [96] and FITEST [4] take as input (1) a set of objectives, (2) a set of fitness functions indicating the degree to which individual objectives have been achieved, and (3) a time budget; it then returns a set of solutions that maximally achieve each objective individually within the given time budget. Therefore, we can apply the algorithms to our problem by carefully defining a set of corresponding objectives and fitness functions.

Based on the problem definition in Section 4.1, we can define that an objective for each key-point is to find a test image that makes a FKP-DNN under test *severely mispredict* the key-point position. Specifically, for a FKP-DNN d, the set of objectives to be achieved by a many-objective search algorithm is $NE(p_{t,i}, \hat{p}_{t,i}) \geq \epsilon$ for all $i = 1, \ldots, k$ where $NE(p_{t,i}, \hat{p}_{t,i})$ is the normalized error of the predicted position $\hat{p}_{t,i}$ with respect to the actual position $p_{t,i}$ for the i-th key-point and ϵ is a small threshold pre-defined by domain experts. Naturally, the set of fitness functions corresponding to the objectives is $NE(p_{t,i}, \hat{p}_{t,i})$ for all $i = 1, \ldots, k$.

While it seems intuitive to define the sets of objectives and fitness functions for many-objective search algorithms, the issue in practice is how to determine the actual positions of key-points in an automated manner; a simulator plays a key role here.

Figure 4.2 shows the overview of the search-based test suite generation process for a FKP-DNN using a simulator. It is composed of four main components: a search engine, a simulator, the DNN under test, and a fitness calculator. The process begins with the search engine generating a set of new input values for image characteristics $\mathbf{ic}$ (e.g., head posture and skin color). The input values are used by the simulator to generate a new test image t as well as the actual positions of key-points $\mathbf{p}_t = \langle p_{t,1}, \ldots, p_{t,k} \rangle$ for all $i = 1, \ldots, k$. The FKP-DNN under test d takes t as input and returns the predicted positions of the key-points $d(t) = \langle \hat{p}_{t,1}, \ldots, \hat{p}_{t,k} \rangle$. Then, the fitness calculator computes the fitness score of t using $NE(p_{t,i}, \hat{p}_{t,i})$ for each $i = 1, \ldots, k$. The fitness score is fed back to the search engine to generate new and better input values over the next iterations. This process continues until either a given time budget runs out or all the objectives are achieved. The process ends with returning a test suite TS such that each $t \in TS$ satisfies $NE(p_{t,i}, \hat{p}_{t,i}) \geq \epsilon$ for some $i = 1, \ldots, k$. In the following subsections, we will explain each of the approach components in more detail.

4.2.1 Search Engine

The search engine drives the whole process based on a meta-heuristic search algorithm. For a given set of objectives with fitness functions and a time budget, it iteratively generates new test inputs for image characteristics ($\mathbf{ic}$) to ultimately provide engineers with the most effective test suite. Search algorithms, such as Random Search (RS), MOSA [96], and FITEST [4], vary in the way they generate new test inputs

based on the fitness results from previous iterations.

For example, MOSA starts with an initial set of randomly generated test cases that forms an initial population; then, it creates new test cases using *crossover* and *mutation* operators that are typically used in Genetic Algorithms (GA) to find better candidates while promoting their diversity. Unlike GA, the *selection* in MOSA is performed by considering both the non-dominance relation and the uncovered objectives. Specifically, for each uncovered objective u, a test case t will have a higher chance of remaining in the next generation if t is non-dominated by the others and is the closest to cover u. To form the final test suite, MOSA uses an *archive* that keeps track of the best test cases that cover individual objectives. FITEST is basically the same as MOSA, except that it dynamically reduces the population size with each iteration, according to the number of uncovered objectives.

Algorithm 1: Pseudo-code of MOSA and FITEST

```
   Input  : Set of Objectives O
   Output : Archive (Test Cases) A
1  Archive A ← ∅
2  Set of Uncovered Objectives U ← O
3  Set of Test Cases P ← initialPopulation(|O|)
4  Set of Covered Objectives C ← calculateObjs(O, P)
5  A ← updateArchive(A, P, C)
6  U ← updateUncoveredObjs(U, C)
7  while not(stoping_condition) do
8  |  Set of Test Cases Q ← generateOffspring(P)
9  |  C ← calculateObjs(O, Q)
10 |  A ← updateArchive(A, Q, C)
11 |  U ← updateUncoveredObjs(U, C)
12 |  P ← getNextGeneration(P ∪ Q, U)
13 end
14 return A
```

More specifically, Algorithm 1 presents the pseudo-code of MOSA and FITEST. It takes as input a set of objectives O and returns an archive A (i.e., a test suite) that aims to maximally achieve individual objectives in O. To do that, the algorithm begins with initializing A, the set of uncovered objectives U, and the population P (lines 1–3). The algorithm then computes the set of covered objectives $C \subseteq O$ by calculating the fitness scores of P for O (line 4), updates A to include test cases from P, which are best at achieving C (line 5), and updates U to remove the covered objectives in C (line 6). Until the stopping criterion is met, the algorithm repeats the followings: (1) generating a set of new test cases Q from P using *crossover* and *mutation* (line 8), (2) updating C, A, and U for Q, as done for P (lines 9–11), and (3) generating the next generation P from the current P and Q considering U using *selection* (line 12). The algorithm ends by returning A. The main difference between MOSA and FITEST is in the *getNextGeneration* function (line 12): MOSA keeps $|P| = |O|$, whereas FITEST reduces $|P|$ as $|U|$ decreases.

To improve the performance of MOSA and FITEST, we can also consider their variants, namely MOSA+ and FITEST+. These variants are identical to the originals, but the crossover strategy is changed to only consider fitness scores for uncovered (not-yet-achieved) objectives to better guide new test cases towards them. In other words, the higher the fitness score for uncovered objectives, the more similar children are to parents. This is done by dynamically updating the distribution index of the Simulated Binary Crossover (SBX), based on the fitness score of parents, for uncovered objectives.

In practice, using an effective and efficient search algorithm, for a given problem, is critical for testing at scale and therefore be applicable in industrial contexts. This is also the case here for generating better test suites for FKP-DNNs.

4.2.2 Simulator

The simulator is one of the key enablers of our approach because it takes as input $\mathbf{ic}_t$ and generates t *labeled* with $\mathbf{p}_t$. It should be able to generate diverse test images by manipulating various $\mathbf{ic}$s, such that there is variation in head posture, gender, skin color, and mouth size. At the same time, it should be able to generate individual test images within reasonable time since this strongly affects the efficiency of the entire search-based process.

To achieve this, one can use 3D modeling tools, such as MakeHuman [25] and Blender [24]. MakeHuman can be used to create parameterized 3D face models with detailed morphological characteristics, such as skin color and hair style. Blender, on the other hand, can be used to manipulate the 3D models according to selected $\mathbf{ic}$ values [15]. Since a 3D model is basically a textured polygonal mesh (i.e., a collection of nodes and edges in 3D) that defines the shape of a polyhedral object, an engineer can mark the actual positions of key-points in the model by selecting the corresponding nodes in the textured mesh. Then, using the simulator, it is easy to generate a 2D image capturing the face labeled with the actual key-point positions.

4.2.3 Fitness Calculator and Fitness Functions

The fitness calculator takes $p_{t,i}$ and $\hat{p}_{t,i}$ and returns $NE(p_{t,i}, \hat{p}_{t,i})$ for each $i = 1, \ldots, k$. As mentioned in Section 4.1, one way to calculate $NE(p_{t,i}, \hat{p}_{t,i})$ is to compute the Euclidean distance between $p_{t,i}$ and $\hat{p}_{t,i}$ and then normalize it using the height or width of the actual face, whichever is larger. Normalization enables the comparison of the error values of different 3D models having different face sizes.

4.3 Empirical Evaluation

In this section, we report on the empirical evaluation of our many-objective, search-based test suite generation approach when applied to an industrial FKP-DNN developed by IEE. We aim to answer the following research questions.

RQ1: *How do alternative many-objective search algorithms fare in terms of test effectiveness?* RQ1 aims to check whether using many-objective search algorithms, such as MOSA and FITEST, is indeed a suitable solution for the problem of test suite generation for FKP-DNNs. To answer this, we investigate how many key-points are severely mispredicted (according to our earlier definition) by test suites that are automatically generated by the search algorithms, and how alternative algorithms compare to each other and to a simple random search.

RQ2*: Can we further distinguish search algorithms using the degree of mispredictions caused by the test suites they generate?* Since RQ1 only considers the number of severely mispredicted key-points, differences in effectiveness across search algorithms may not appear clearly and completely. For example, two test suites generated by different algorithms may cause the same number of severely mispredicted key-points. However, since the level of risk entailed by mispredictions may be proportional to the distance between the actual and predicted key-points, a test suite with a higher degree of mispredictions may be

more amenable to risk analysis, depending on the application context. Based on this, RQ2 compares how *severely* key-points are mispredicted by test suites generated across different search algorithms.

***RQ3**: Can we explain individual key-point mispredictions in terms of image characteristics?* In addition to the cost-effective, automated testing of FKP-DNNs, engineers need to understand why severe mispredictions occur for individual key-points. This is critical for engineers to analyze and address the root causes of mispredictions, or, when the latter is not possible, at the very least assess the risks. For example, if engineers know that a certain key-point is significantly mispredicted for a specific head posture range, they can generate more test images in that range and retrain the DNN to improve its prediction accuracy. To this end, RQ3 aims to investigate whether it is possible to provide accurate and interpretable explanations of mispredictions based on image characteristics used by the simulator to generate test images.

4.3.1 Case Study Design

We use a proprietary FKP-DNN, denoted IEE-DNN, developed by IEE to build a driver's gaze detection component for autonomous vehicles. To enable simulation-based testing, we also use an in-house simulator, namely IEE-SIM, developed by IEE to generate large numbers of labeled facial images to train the IEE-DNN.

DNN

IEE-DNN takes as input a 256x256 pixel image; it returns the predicted positions of 27 facial key-points in the input image. Figure 4.1 depicts the actual positions of the 27 key-points for a sample image.

The IEE-DNN is based on the stacked hourglass architecture [93] with an adaptive wing loss function [135]. Specifically, it consists of two hourglass modules, each of which contains four residual modules for downsampling and another four for upsampling. It is trained on 18,120 syntactic images generated by the IEE-SIM. Against an independent set of 2,738 syntactic test images, the IEE-DNN achieved a low NME value of 0.018, on average. Given that the threshold for labelling mispredictions as severe for individual key-points is 0.05, such NME value implies that, if we just consider the average prediction error for all key-points, the IEE-DNN is sufficiently accurate according to the test set. Note that our approach is independent from any specific DNN architecture as it does not require any DNN internal information.

Simulator

IEE-SIM takes as input various image characteristics, such as head posture, light intensity, and the position of a (virtual) camera, and returns a corresponding facial image with the actual positions of the 27 key-points.

The IEE-SIM is based on MakeHuman [25] and Blender [24]. To decrease its execution time, IEE engineers carefully designed many 3D face models in advance, by considering the diversity in skin colors and the shapes and sizes of faces, mouths, and noses. By doing this, we can quickly generate an image by specifying a pre-defined 3D model ID to use instead of dynamically generating the 3D models from scratch at run time. As a result, the average execution time for generating one labeled image is around 6 seconds on an iMac (3GHz 6-Core Intel i5 CPU, 40GB memory, and Radeon Pro 570X 4GB graphic card).

The main concern of IEE is to verify the accuracy of the IEE-DNN for diverse head postures and drivers. Therefore, we manipulate four feature values when varying image characteristics: roll, pitch, and yaw values for controlling head posture and the 3D model ID for indirectly the face features to be used. The ranges of the roll, pitch, and yaw values are limited between -30° and +30° to mimic realistic ways in which drivers position their head. For 3D models, we use the subset of 10 different models that IEE considered to be of main interest.

4.3.2 RQ1: Effectiveness of Test Suites

Setup

To answer RQ1, we generate a test suite using our approach for a fixed time budget and measure the *Effectiveness Score* (*ES*) of the test suite, defined as the proportion of key-points that are severely mispredicted by the IEE-DNN according to the test suite over the total number of key-points. *ES* ranges between 0 and 1, where higher values, when possible, are desirable.

To better understand how *ES* varies across different search algorithms, we compare MOSA [96], FITEST [4], and their variants MOSA+ and FITEST+ (see Section 4.2.1). We use Random Search (RS) as a baseline. RS randomly generates a test suite for each iteration and keeps the best until the search process ends; RS provides insights into how easy the search problem is and helps us assess if other, more complex search algorithms are indeed necessary.

One might consider using NSGA-III as another baseline that tries to achieve many objectives collectively. However, it requires addressing the problem of how to encode test suites into chromosomes, which is a research subject that is beyond the scope of our chapter.

For MOSA, FITEST, and their variants, the initial population size is the number of objectives. To be consistent, we set the number of randomly generated test cases at each iteration of RS to be the number of objectives. For the other parameters, such as mutation and crossover rates in MOSA, FITEST, and their variants, we use the default values recommended in the original studies.

To account for randomness in search algorithms, we repeat the experiment 20 times. For each run, we use the same time budget of two hours for all search algorithms, based on our preliminary evaluation showing that two hours are enough to converge. We apply the non-parametric Mann–Whitney U test [83] to assess the statistical significance of the difference in *ES* between algorithms. We also measure Vargha and Delaney's $\hat{A}_{AB}$ [132] to capture the effect size of the difference.

Results

Figure 4.3 depicts the differences in *ES* across search algorithms. RS only achieves $ES = 0.41$ on average across 20 runs. Though such ES value may seem a priori high, one must recall that it cannot be interpreted as a failure rate in the normal sense since certain key-points are impossible to predict, e.g., some key-points are hidden by a shadow. In general, it is to be expected that such DNNs will have inherent limitations that cannot be addressed by retraining or any other means. It is up to the system using a DNN to adequately address such limitations, for example by not using key-point predictions in shadowed areas and relying on the other key-points for decisions. Therefore, the purpose of *ES* values is only to compare the test effectiveness of different algorithms and not to assess the accuracy of the DNN under test. In contrast to RS, MOSA, MOSA+, and FITEST+ reach $ES = 1$ in at least one run, meaning that these

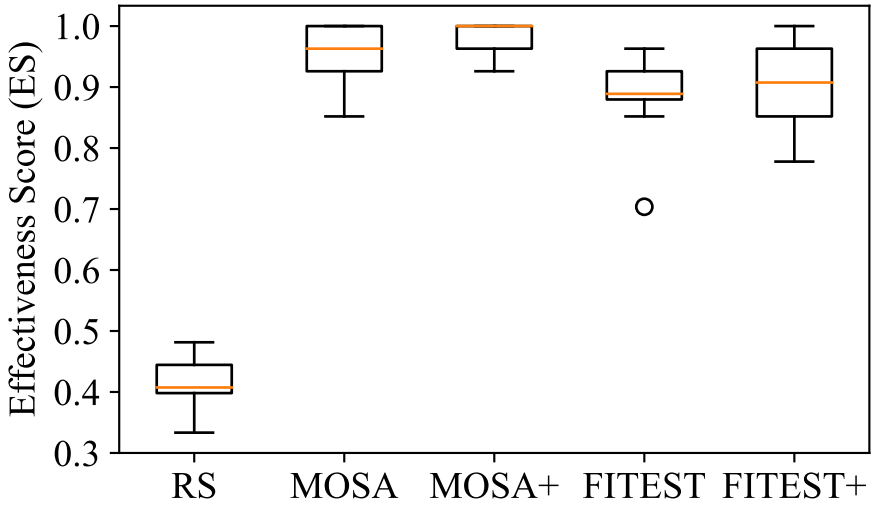


Figure 4.3: Test effectiveness for different search algorithms

Table 4.1: Statistical Analysis Results

Pair		*ES*		*MS*	
A	B	p-value	$\hat{A}_{AB}$	p-value	$\hat{A}_{AB}$
MOSA	RS	0.000	1.00	0.000	0.90
MOSA+	RS	0.000	1.00	0.000	0.90
FITEST	RS	0.000	1.00	0.000	0.88
FITEST+	RS	0.000	1.00	0.000	0.87
MOSA	FITEST	0.000	0.84	0.570	0.54
MOSA+	FITEST	0.000	0.95	0.594	0.54
FITEST+	FITEST	0.750	0.53	0.052	0.51
MOSA	FITEST+	0.005	0.76	0.177	0.55
MOSA+	FITEST+	0.000	0.86	0.009	0.56
MOSA+	MOSA	0.109	0.64	0.780	0.50

algorithms could generate test suites that cause the IEE-DNN to severely mispredict all 27 key-points. Across 20 runs, the many-objective search algorithms achieve $ES > 0.93$ on average, implying that our approach seems indeed effective at generating test suites that cause severe mispredictions for many key-points.

Table 4.1 shows the results of statistical comparisons between different search algorithms. Under the *Pair* column, sub-columns *A* and *B* indicate the two search algorithms being compared. Under the *ES* column, sub-columns *p-value* and $\hat{A}_{AB}$ indicate the statistical significance and the effect size, respectively, when comparing *ES* distributions between *A* and *B*.

At a level of significance $\alpha = 0.01$, we can see that MOSA is significantly better than FITEST, with large effect size (0.84). Comparing MOSA+ and FITEST+ shows the same result, with an even larger effect size (0.86). This implies that (dynamically) reducing the population size at each iteration significantly degrades the effectiveness of the search algorithms for test suite generation, as it decreases their ability to explore the search space. When comparing MOSA (and FITEST) and MOSA+ (and FITEST+), their differences in *ES* are statistically insignificant, implying that dynamically controlling the

similarity between parents and children in crossover does not significantly improve effectiveness. In RQ2, we will further compare the many-objective search algorithms by considering the degree of misprediction severity across the test suites they generate.

The answer to RQ1 is that our approach is effective in generating test suites that cause IEE-DNN—which is already rather accurate (NME = 0.018 for the test dataset) as one might expect from an industrial model—to severely mispredict more than 93% of all key-points on average. This number is also much higher than that obtained with random search (41%). While MOSA and MOSA+ are significantly better than FITEST and FITEST+ in terms of *ES*, there is no significant difference between MOSA and MOSA+, which will be investigated further in RQ2.

4.3.3 RQ2: Misprediction Severity for Individual Key-Points

Setup

To answer RQ2, following the same procedure as for RQ1, we measure the *Misprediction Severity* (*MS*) of a test suite for each key-point, defined as the maximum NE value observed when running the test suite. As for NE, *MS* ranges between 0 and 1, where 1 implies the maximum prediction error.

Since RQ2 aims to further distinguish the search algorithms by considering the *MS* values of test suites they generated in RQ1, we use the test suites from RQ1 and report the average *MS* of individual key-points for each algorithm. We apply the non-parametric Wilcoxon signed-rank test to statistically compare *MS* distributions for each key-point between search algorithms, and measure Vargha and Delaney's $\hat{A}_{AB}$ to capture the effect size of the difference.

Results

Figure 4.4 shows average *MS* values of the test suites generated by different search algorithms, for individual key-points. For example, we can see that the average *MS* value with MOSA+ for the 26th key-point (i.e., KP26) is around 0.8.

Comparing MOSA, FITEST, and their variants, we can see that the overall patterns shown in the radar chart are similar. This implies that the many-objective test suite generation algorithms yield similar patterns in terms of which individual key-points tend to get higher *MS* values. To further assess the difference between them, we need to check the results of the statistical tests.

Table 4.1 shows the results of statistical comparisons between different search algorithms. Under the *MS* column, sub-columns p-*value* and $\hat{A}_{AB}$ indicate the statistical significance and the effect size, respectively, when comparing the *MS* distributions of two algorithms under columns *A* and *B*.

In Table 4.1, with $\alpha = 0.01$, there is no statistically significant difference in *MS* between MOSA and MOSA+, and between FITEST and FITEST+. This implies that, consistent with RQ1, dynamically adjusting the distribution index in crossover does not increase misprediction severity for individual key-points.

As for MOSA and FITEST, which were significantly different with respect to *ES* in RQ1, they are no longer significantly different regarding *MS*. This inconsistency between RQ1 and RQ2 happens because, even though MOSA (and MOSA+) is better than FITEST (and FITEST+) in making the NE values of more key-points exceed $\epsilon = 0.05$, FITEST (and FITEST+) yields higher NE values than MOSA (and

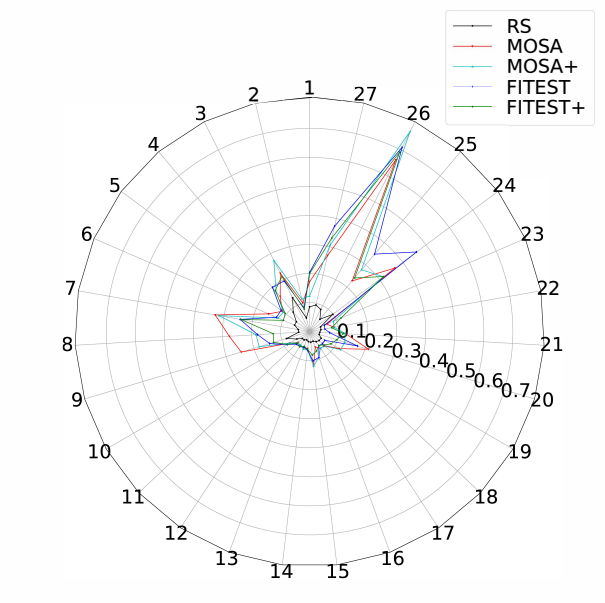


Figure 4.4: Misprediction Severity (*MS*) for individual key-points for different search algorithms

MOSA+) for some key-points. When it comes to MOSA+ and FITEST+, MOSA+ is significantly better than FITEST+ with a small effect size (0.56). The results between MOSA and FITEST, and MOSA+ and FITEST+, imply that dynamically reducing the population size during the search increases the degree of mispredictions for some key-points, but the overall impact is limited.

Interestingly, Figure 4.4 shows that some key-points (i.e., KP7, KP24, KP25, KP26, and KP27) are more severely mispredicted than the others. A detailed analysis found two distinct reasons that together affect the accuracy of the DNN: the under-representation of some key-points in the training data and a large variation in the shape and size of the mouth across different 3D models. For KP7, we found that it is only included in 79% of the training data. One possible explanation is that, as shown in Figure 4.1, KP7 can easily become invisible depending on the head posture. Interestingly, KP9, which has a symmetrical position to KP7 in the face, is included in a larger proportion of the training data (84%). Such situations typically happen when the training set is not built in a systematic fashion. This implies that key-points that are under-represented in training data, because they happen not to be visible on numerous occasions, are more likely to be severely mispredicted. On the other hand, KP24, KP25, KP26, and KP27 are more severely mispredicted than the others, even though they are included in most of the training images (more than 92%). These four key-points are severely mispredicted because they are related to the mouth, which shows the largest variation among face features; the mouth can be opened and closed, and is larger than the eyes. Therefore, the positions of the mouth key-points vary more than other key-points in the face, even when the head posture is fixed. As a result, learning the position of the mouth key-points is more difficult than that for other key-points. This implies that, during training, we need to focus more on certain key-points whose actual positions can vary more than the others across images.

The answer to RQ2 is that, by additionally considering *MS*, we cannot further distinguish MOSA and FITEST, and their variants. Instead, through a detailed analysis of the most severely mispredicted key-points, we identified important insights to prevent severe mispredictions: (1) since some key-points may not actually be visible in an image, depending on image characteristics (e.g., head posture), for each key-point, it is important to ensure that the training data contains enough images where the key-point is visible, (2) more training is required for certain key-points, whose positions tend to vary significantly more across the face and which are harder to predict than the others. Such observations can obviously generalize to other types of images and key-points.

4.3.4 RQ3: Explaining Mispredictions

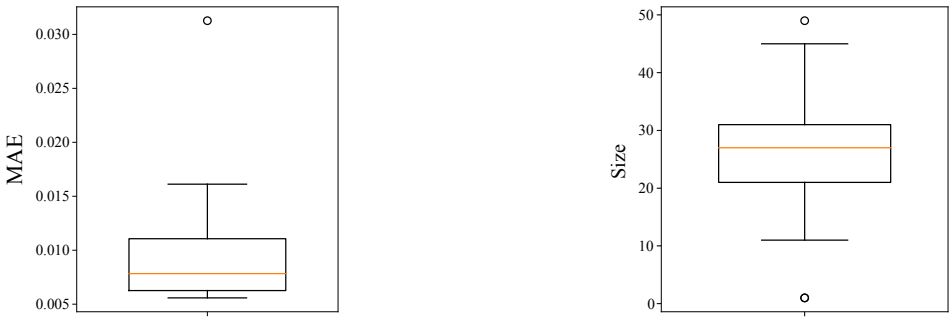
Setup

To answer RQ3, decision trees [14] are learnt to infer how the NE (normalized prediction error) of the FKP-DNN for individual key-points relate to image characteristics used by the simulator to generate test images. We use decision trees because they are easy to interpret due to their hierarchical decision-making process [92], though alternative forms of rule-based learning could be considered as well. Interpretability is essential for engineers to assess the risks associated with a DNN in the context of a given application and to devise ways to improve the DNN, for example, through additional training data. Specifically, we build a decision tree for each key-point to identify conditions describing how NE varies according to input variables, i.e., roll, pitch, yaw, and 3D model ID. Since our test suite generation approach generates many test images specified by roll, pitch, yaw, and 3D model, and given that NE values are calculated for individual key-points during the search process, we can use this information as a set of observations for building decision trees and explain mispredictions. To mimic a practical scenario in which engineers use our approach overnight (e.g., ten hours), we use the information collected from five random runs (i.e., equivalent to ten hours) of MOSA+ (i.e., the most effective search algorithm according to RQ1 and RQ2). Since the target variable (i.e., NE) is continuous, we use regression trees to explain and predict the degree of mispredictions. Specifically, we use REPTree (i.e., fast tree learner using information-gain and reduced-error pruning) implemented in Weka [141].

To evaluate the (predictive) accuracy of generated regression trees, we measure the Mean Absolute Error (MAE) using 10-fold cross validation. Note that, to provide interpretable explanations, the simplicity of the resulting regression trees is just as important as its accuracy. In particular, as the number of nodes in a regression tree increases, it becomes increasingly difficult for engineers to interpret the results. As a trade-off between accuracy and simplicity, we set the minimum number of observations per leaf node to 40, based on preliminary experiments. For the other parameters for REPTree, we use default values provided by Weka.

Results

Figure 4.5 shows the MAE and size (i.e., number of nodes) distributions for all 27 trees built based on 3854 test images obtained from five runs of MOSA+. The average MAE is 0.01 and the average size is 25.7. We can see that the trees are accurate as there is an average difference of 0.01 between the actual



(a) Error (b) Size

Figure 4.5: Regression tree accuracy and size for all key-points

Table 4.2: Representative rules derived from the decision tree for KP26 (M: Model-ID, P: Pitch, R: Roll, Y: Yaw)

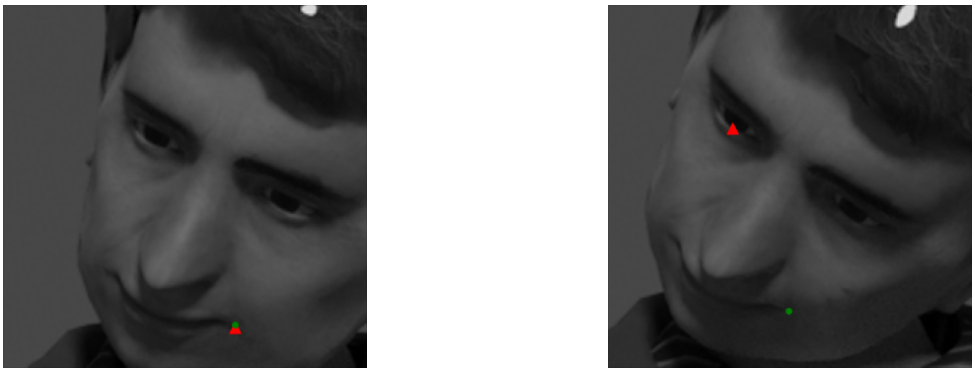
ic-condition	*NE*
$M = 9 \wedge P < 18.41$	0.04
$M = 9 \wedge P \geq 18.41 \wedge R < -22.31 \wedge Y < 17.06$	0.26
$M = 9 \wedge P \geq 18.41 \wedge R < -22.31 \wedge 17.06 \leq Y < 19$	0.71
$M = 9 \wedge P \geq 18.41 \wedge R < -22.31 \wedge Y \geq 19$	0.36

and predicted NE values, which is far below the threshold that IEE finds acceptable (0.05). The trees are also of reasonable size and therefore easy to interpret as they lead to simple rules, as further shown below.

Table 4.2 shows some representative rules derived from the tree generated for KP26, the most severely mispredicted key-point in RQ2. All the remaining rules and the trees for the other key-points are available in the supporting material [2]. In Table 4.2, column **ic**-*condition* shows conditions on values of the image characteristics, i.e., Roll (R), Pitch (P), Yaw (Y), and 3D Model ID (M); column NE shows the average NE value for all test images satisfying the condition. For example, the first row from the table means that, for test images such that 3D model ID is 9 and pitch is less than 18.41, the average NE is 0.04. Recall that each 3D model ID corresponds to implicit facial characteristics. For example, for model ID 9, skin color is brown, face structure is broad, nose type is aquiline, and mouth structure is uneven.

Using such conditions, engineers can easily identify when the FKP-DNN leads to severe mispredictions, for individual key-points. For example, by comparing the first and third conditions in Table 4.2, we can see that, for the same 3D model, changing the head posture toward specific ranges leads to a significant increase of the prediction error of the IEE-DNN for KP26. To better visualize the example above, we present two test images, in Figures 4.6a and 4.6b, satisfying the first and third conditions, respectively. In each image, the green dots and red triangle indicate the actual and predicted positions of KP26. While there is a small prediction error ($NE = 0.013$) in Figure 4.6a, it gets much larger ($NE = 0.89$) when turning the head further down and to the right, as shown in Figure 4.6b. Engineers can further investigate the root causes of severe mispredictions using targeted additional images. For example, based on Figure 4.6b, we generated a targeted image that differs only in the direction of the shadow (by changing the position of the light source in the 3D model) and could confirm that the shadow was indeed

[2] All the decision trees are available on https://figshare.com/s/0433751b88282b07ea12. Unfortunately, the full replication package could not be made publicly available since the IEE-SIM and the IEE-DNN are proprietary assets.



(a) An image satisfying the first condition in Table 4.2 (b) An image satisfying the third condition in Table 4.2

Figure 4.6: Test images satisfying different conditions in the regression tree for KP26. Actual and predicted positions are shown by green-circle and red-triangle dots, respectively.

the root cause of the large NE value.

Knowing under what conditions severe mispredictions are occurring can help engineers in two ways. First, it helps to assess the risks associated with individual key-points for specific conditions, in the context of a specific application. For example, if the head posture can be constrained to $P < 18.41$ in the context of a certain application of the FKP-DNN, the risk of severe misprediction for KP26 is reduced according to the conditions in Table 4.2. Second, it enables the generation of specific test images, using the simulator, that are expected to cause particularly severe mispredictions and can be used for retraining the DNN. For example, from the conditions in Table 4.2, we can generate new images satisfying $M = 9 \wedge P \geq 18.41 \wedge R < -22.31$, which will likely lead to high NE values, and can be used to help retrain and improve the DNN by augmenting its training data set.

The answer to RQ3 is that, by building regression trees using the information that is produced by executing our test generation approach, we can largely explain the variance in individual key-point mispredictions in terms of image characteristics that are controllable in the simulator. These conditions can help engineers assess risks for individual key-points and generate specific images for retraining of the DNN in a way that is more likely to have an impact on its accuracy.

4.3.5 Threats to Validity

In RQ1, we have used a threshold ($\epsilon = 0.05$) for deciding if a key-point is severely mispredicted or not. This may affect the results. However, this threshold value was set by IEE domain experts, accounting for the specific application of their DNN. Furthermore, we checked that using other small threshold values does not severely affect overall trends but only slightly affect ES values.

We have used 3D face models with various facial characteristics shared by IEE. IEE engineers have prepared the 3D models by marking key-points' actual positions manually. The manual marking can be erroneous and such error can affect the quality of data generated for training and testing the DNN. To mitigate such threats, IEE engineers have done extensive testing for each 3D model and carefully validated the generated data.

Though we focused, in our case study, on one FKP-DNN developed by IEE, it is representative of what one can find in industry [58, 20, 59], in terms of prediction accuracy, inputs and outputs, and

training procedure. Though there are no publicly available KP-DNNs coupled with a simulator, additional industrial case studies are, however, required to improve the generalizability of our results.

Since we focused on syntactic images generated by a simulator, the transferability of our results from simulation to real-world facial key-points detection is not in the scope of this work. Nevertheless, we want to note that [52] already provided an experimental methodology to quantify such transferability and reported that simulator-generated data can be a reliable substitute to real-world data for the purpose of DNN testing.

4.3.6 Lessons Learned

Lesson 1: Automated test suite generation is indeed useful in practice.

Although we have presented here the evaluation results using the latest IEE-DNN version, our approach has been continuously applied to previous versions during its development process. Indeed, test suites automatically generated by our approach and the analysis of testing results for mispredicted key-points have helped assess and improve IEE-DNNs. In particular, our approach led IEE to augment their training dataset (e.g., diversity and sample size) and thus improve the generalization of the IEE-DNN against various inputs. In summary, based on the results, IEE has been (1) continuously enriching their dataset by adding more training images from diverse 3D face models and (2) improving the IEE-DNN's architecture by doubling the number of hidden layers to drastically increase its accuracy.

Our approach has been also used for improving the simulator. For example, during a detailed analysis of the testing results, we revealed a critical issue: the labeled key-point positions on test images, generated by the IEE-SIM, were not accurate and, after analysis, engineers realized there was a misalignment between the texture and nodes of the 3D mesh that are used to define the actual position of key-points.

IEE also plans to use our approach in various ways. Since the test suites generated by our approach cause severe mispredictions for many key-points, retraining the DNN using these test suites—given that the DNN architecture is now adequate—is expected to help improve the accuracy of the DNN. We will further investigate to what extent such accuracy can be improved in our future work.

Lesson 2: Understanding mispredictions is critical.

Though explaining DNNs [109] is important in many circumstances, the complex structure of such models makes it challenging. Instead of analyzing the internals of the IEE-DNN, we simply built decision trees for inferring conditions characterizing key-point mispredictions in terms of input variables, i.e., roll, pitch, yaw, and 3D model ID. Even such analysis cannot directly explain the behavior of the IEE-DNN in detail, knowing such conditions brings several benefits to IEE. First, it helps them demonstrate the robustness of the DNN under certain conditions which, for generated test images, do not lead to severe mispredictions. If those conditions match the conditions of the intended application, then engineers can conclude that the DNN is likely to be safe for use. Second, it facilitates the investigation of the root causes of mispredictions. This is critical because, similar to the severe misprediction in Figure 4.6b discussed in § 4.3.4, some mispredictions are not avoidable and can only be addressed by engineering the system using the DNN for robustness (e.g., by identifying shadowed areas and not using predictions for the key-points in these areas). Such finding led IEE to better target their development resources to improve the driver's gaze detection system rather than just focusing on the IEE-DNN itself.

Lesson 3: Simulation-based testing brings key benefits.

As discussed in Section 4.2.2, a simulator is one of the key enablers of our approach as it generates *labeled* test images in a controlled manner. However, the simulator can also be a limiter as test suites generated by our approach depend on what the simulator can generate. Ideally, one would like access to a high-fidelity and configurable simulator that is able to generate diverse images, accounting for all major factors that affect the behavior of the DNN under test. For example, the IEE-SIM has been developed to support the configuration of various image characteristics, such as head posture, face size, and skin color, that are essential because of their effect on facial key-point detection. Thanks to the simulator, we can generate as many different test images as we need, with known key-points, and drive effectively automated test generation. Given the usefulness of our approach, the cost of having a simulator, with sufficient fidelity and configurability, benefits the development and testing of KP-DNNs by enabling efficient automation.

4.4 Related Work

Many automated techniques are available in the literature for testing DNNs. One type of automatic test data generation techniques is to generate adversarial examples [146] that are imperceptible for humans but cause the DNN under test to misbehave. Guo et al. [48] proposed DLFuzz, an approach that iteratively applies small perturbations to original images to maximize neuron coverage (i.e., the rate of activated neurons) and prediction differences between the original and synthesized images. Zhou et al. [152] and Kong et al. [68] proposed DeepBillboard and PhysGAN, respectively, two similar approaches that generate adversarial images that can be placed on drive-by billboards, both digitally or physically, to induce failures in DNN-based automated driving systems. Wicker et al. [140] presented an approach to generate adversarial images using feature extraction techniques, such as Scale Invariant Feature Transform (SIFT), to extract features from original images. Rozsa et al. [107] introduced a Fast Flipping Attribute (FFA) technique to effectively generate adversarial images for DNNs detecting facial attributes (e.g., male or female). However, in contrast to our objective of verifying the safety of DNN predictions for a large variety of plausible test inputs, adversarial examples mainly target security attacks where an attacker intentionally introduces human-imperceptible changes (i.e., attacks) to cause the DNN to generate incorrect predictions.

Another important line of work is to generate new test data from already labeled data (e.g., training data) by applying label-preserving changes to avoid labeling problems for newly generated test inputs. Pei et al. [99] proposed DeepXplore, an approach that generates label-preserving test images to maximize both neuron coverage and differential behaviors of multiple DNNs for the generated images. Tian et al. [125] introduced DeepTest, an approach that synthesizes label-preserving test images by applying affine transformations and effect filters, such as rain and snow, to original images in order to maximize neuron coverage. Zhang et al. [148] presented DeepRoad, an approach that uses Generative Adversarial Networks (GANs), instead of simple transformations and effect filters, to generate more realistic images. Du et al. [33] proposed DeepCruiser, an approach that generates label-preserving sequences of test data to test stateful deep learning systems, based on Recurrent Neural Networks (RNNs) using special coverage criteria for RNNs converted to Markov Decision Process (MDP). Recently, Xie et al. [144] presented DeepHunter, an extensible coverage-guided testing framework extending DeepTest to further utilize multiple coverage criteria, with more label-preserving transformation strategies. While these works

enable engineers to generate more realistic test data from existing data, generating label-preserving test images inherently limits the search space when the objective is to identify critical situations in the most comprehensive way possible. Furthermore, when relying on a simulator, labeling is not a critical issue.

To overcome the limitation of label-preserving test data generation, simulation-based testing is increasingly used for testing DNNs, especially in the context of DNN-based automated driving systems. Gambi et al. [41] presented AsFault, a search-based approach to generates different types of road topologies in a simulated environment to test the lane keeping functionality of self-driving DNNs. Tuncali et al. [127] introduced Sim-ATAV, a framework for testing closed-loop behaviors of DNN-based systems in simulated environments using requirement falsification methods. Riccio and Tonella [103] presented DeepJanus, an approach to generate pairs of similar test inputs, using search-based testing with simulations, that cause the DNN under test to misbehave for one test input but not for the other. Although the solutions in this category successfully used simulation to generate critical test data, they are inadequate for the test suite generation of FKP-DNNs, because they do not consider many independent outputs individually. Therefore, even when it seems infeasible to find a critical image for a certain key-point, these solutions cannot dynamically redistribute computational resources to to other key-points.

In summary, existing work does not address the fundamental and specific challenges of test suite generation for KP-DNNs, as described in Section 4.1. To this end, we propose a solution, based on many-objective optimization, that automatically generates test suites whose objective is to cause KP-DNNs to severely mispredict the positions of individual key-points and then use machine learning to explain such mispredictions.

4.5 Conclusion

In this chapter, we formalize the problem definition of KP-DNN testing and present an approach to automatically generate test data for KP-DNNs with many independent outputs, a common situation in many applications. We empirically compare state-of-the-art, many-objective search algorithms and their variants tailored for test suite generation. We find MOSA+ to be significantly more effective than random search (baseline) and other many-objective search algorithms, e.g., FITEST, with large effect sizes. We also observe that our approach can generate test suites to severely mispredict more than 93% of all key-points on average, while random search, as a comparison, can do so for 41% of them. We further investigate and demonstrate a way, based on regression trees, to learn the conditions, in terms of image characteristics, that cause severe mispredictions for individual key-points. These conditions are essential to engineers to assess the risks associated with using a DNN and to generate new images for DNN retraining, when possible.

Chapter 5

Efficient Online Testing for DNN-Enabled Systems using Surrogate-Assisted and Many-Objective Optimization

In this chapter, we present a novel approach, called SAMOTA (Surrogate-Assisted Many-Objective Testing Approach), that combines two distinct techniques: (1) *many-objective search* [96, 4] to effectively achieve many independent objectives (i.e., causing safety violations) within a limited time budget and (2) *surrogate-assisted optimization* [61] to efficiently search for critical test data using surrogate models that mimic the simulator, to a certain extent, but are computationally much less expensive. In particular, following state-of-the-art surrogate-assisted optimization algorithms [134, 145, 78], SAMOTA uses two search phases: *global search* (with global surrogate models) that *explores* the search space by capturing the global outline of the fitness landscape using global surrogate models, and *local search* (with local surrogate models) that *exploits* the local details around promising areas found by the global search. We also improve the local search performance using a novel, clustering-based approach that generates one local surrogate model for test data belonging to the same promising area.

Though SAMOTA can be applied to any DNN-enabled systems (DES) that should be verified with online testing, we evaluated the efficiency and effectiveness of the approach in the context of ADS. Specifically, we use CARLA [30], a high-fidelity driving simulator, and Pylot [46], an advanced DNN-enabled ADS (DADS) composed of multiple DNNs capable of various tasks, such as traffic light detection, traffic sign detection, object tracking, and object classification. More than 300 computing hours of experimental results show that SAMOTA is significantly more effective and efficient at detecting unknown safety violations than state-of-the-art many-objective test suite generation algorithms and random search, within the same time budget.

The contributions of this chapter are summarized as follows:

- SAMOTA, a novel approach to automatically and efficiently generate test data for online testing by carefully combining many-objective search and surrogate-assisted optimization;

- An extensive empirical evaluation of SAMOTA, in terms of efficiency and effectiveness, and its comparison with state-of-the-art alternatives and random search;

- a publicly available replication package of the evaluation, including the implementation of SAMOTA (see Section 5.6).

Significance In many cyber-physical domains, it is clearly important to identify potential safety violations in a DES through online testing, involving a high-fidelity simulator in the loop, especially when there are complex interactions between the system and its environment. SAMOTA provides an efficient and effective online testing approach using surrogate models while considering many safety requirements at the same time. It is a practical enabler for the online testing of complex DES in realistic contexts. Our research is also an important step towards the scalable testing of DES, a very active research area in software testing.

Chapter Structure The rest of the chapter is organized as follows. Section 5.1 formalizes the problem of test suite generation for the online testing of DES. Section 5.2 describes our approach. Section 5.3 evaluates our approach in the context of ADS. Section 5.4 positions our work with respect to related work. Section 5.5 concludes the chapter. Section 5.6 provides the details on the replication package.

5.1 Problem Definition

In this section, we provide a general but precise problem description regarding test suite generation for DES in the context of online testing. We use DADS (DNN-enabled ADS) as an example of DES, but the description can easily be generalized to all DES.

In online testing, the DADS under test are embedded into a driving environment, often in a loop with a simulator due to the high cost and risk associated with the real-world testing of vehicles. Using a simulator also enables the generation of various driving scenarios using the simulator's controllable attributes. In a simulator, the DADS receives sensor data (e.g., an image capturing a driving scene) generated by the simulator and produces control commands (e.g., steering angle, throttle, and brake) to drive the ego vehicle. Since the control commands actually drive the ego vehicle being simulated in online testing, predictions generated by the DNNs of the DADS at time t impact the sensor data to be generated after t. Therefore, it is essential to run the simulator for a specific driving scenario and check if a safety violation occurs. The goal of online test suite generation for DADS is to generate a minimal set of driving scenarios that cause the system under test to violate as many safety requirements as possible.

More specifically, let s be a driving scenario that defines the road topology, weather condition, and the trajectory of other mobile objects (e.g., other vehicles and pedestrians) in a virtual environment. The detailed definition of s (i.e., the definition of test input space) can vary depending on the configurable attributes of the simulator. By embedding a DADS d into a simulator and running it for s, at time t, the simulator generates d's sensor data i_t, such as an image capturing the driving scene (status) $\mathbf{v}_{s,t}$ taken by the front-facing camera mounted on the ego vehicle's dashboard. By taking i_t, d produces controls $d(i_t) = (st_t, ac_t, br_t)$ where st_t, ac_t, and br_t represent steering, acceleration, and braking commands, respectively. The simulator then updates $\mathbf{v}_{s,t+1}$ by taking into account $d(i_t)$. For each $\mathbf{v}_{s,t}$, we can verify if a safety requirement is violated or not. For example, regarding the safety requirement of lane-keeping

(i.e., the ego vehicle should be in the center of the lane), we can measure the distance of the ego vehicle from the center of the lane in $\mathbf{v}_{s,t}$. If the distance is larger than a certain threshold, the safety requirement is violated. In general, let R be a set of safety requirements and $r(\mathbf{v}_{s,t})$ be the degree of the violation for a safety requirement $r \in R$ in $\mathbf{v}_{s,t}$. We say that d violates r in s if $r(\mathbf{v}_{s,t}) > \epsilon_r$ for any t during the simulation time of s, where ϵ_r is a threshold for r. Though that may not be possible, test suite generation attempts to generate a minimal set of test scenarios TS that satisfies $r(\mathbf{v}_{s,t}) > \epsilon_r$ for all $r \in R$ for some test scenarios $s \in TS$.

Test suite generation for DADS in online testing entails several challenges. First, the test input space is too large to be exhaustively explored because there are many attributes having many options that can be selected for generating a certain scenario. For example, road type has multiple options, such as straight, curved, and cross junction. Second, there are many safety requirements, usually independent of each other. For example, complying with traffic lights is independent from keeping the center of a lane. This further increases the complexity of the problem. Third, running a simulator for a scenario is often time-consuming due to the intensive computations required to dynamically update mobile objects and render driving scenes in virtual worlds, with high fidelity; for example, in our case study, one scenario execution in online testing takes 5-10 minutes. Fourth, depending upon the accuracy of the system under test, it may be infeasible to find a scenario causing violations for some safety requirements. Considering a limited time budget, if such infeasibility is observed at run time, it is essential to dynamically and efficiently distribute computation resources to the other safety requirements. Last but not least, DNNs in DADS are often developed by a third party, with expertise in ML, who does not provide access to internal information of the DNNs. Therefore, online testing should be conducted in a black-box manner without relying on internal information.

To address the challenges mentioned above, as detailed in Section 5.2, we suggest combining two distinct techniques: (1) many-objective search algorithms to effectively achieve many independent objectives (i.e., causing safety violations) within a limited time budget and (2) surrogate models that mimic the simulator, to the extent possible, while being computationally much less expensive. Furthermore, since it is black-box, this approach is DNN agnostic, e.g., it does not make any assumptions about the DNN architecture.

5.2 Surrogate-Assisted Many-Objective Search for Test Suite generation

This section provides a solution to the problem of test suite generation for DADS, described in Section 5.1, by combining many-objective search and surrogate models. In the following subsections, we first describe how many-objective search can be used for the problem of test suite generation for DADS. We then present our novel algorithm for surrogate-assisted many-objective search.

5.2.1 Test Suite Generation using Many-Objective Search

As described in background chapter (section 2.1), MOSA [96] and FITEST [4] have been introduced in the context of software testing to maximally cover individual test targets (e.g., branches). Therefore, we can apply the algorithms to our problem by carefully defining a set of corresponding test targets (objectives) and fitness functions.

Based on the problem definition in Section 5.1, we can define test objectives as violations of safety requirements. Specifically, for a set of safety requirements $R = \{r_1, \ldots, r_n\}$ and a DADS d, the objectives to be achieved by a many-objective search algorithm are $r_i(\mathbf{v}_{s,t}) > \epsilon_{r_i}$ for all $i = 1, \ldots, n$ where $r_i(\mathbf{v}_{s,t})$ is the degree of violation of d for a safety requirement r_i in scenario s at time t and ϵ_{r_i} is a violation threshold pre-defined by domain experts for each requirement r_i. In other words, the set of objectives is defined as $O = \{o_1, \ldots, o_n\}$ where o_i is $r_i(\mathbf{v}_{s,t}) > \epsilon_{r_i}$ for $i = 1, \ldots, n$.

Using the set of objectives, we can apply MOSA or FITEST whose pseudo-code is presented in Algorithm 2. It takes as input a set of objectives $O = \{o_1, \ldots, o_n\}$, a population size p_s, and a set of thresholds $E = \{\epsilon_{r_1}, \ldots, \epsilon_{r_n}\}$; it returns an archive (i.e., a test suite) A that aims to maximally achieve individual objectives in O.

Algorithm 2: Many-Objective Search for Test Suite Generation

Input : Set of Objectives O
Population Size p_s
Set of Error Threshold E
Output : Archive A

```
1  Archive A ← ∅
2  Set of Uncovered Objectives U ← O
3  Set of Test Cases P ← initialPopulation(p_s)
4  while not (stopping-condition) do
5  |  Set of Test Cases Q ← generateOffspring(P)
6  |  Set of Test Cases W ← calculateFitnessSim(P ∪ Q)
7  |  A, U ← updateArchive(A, W, E, O)
8  |  P ← generateNextGen(W, U)
9  end
10 return A
```

The algorithm begins with initializing A, a set of uncovered objectives U, and a set of test cases (i.e., test scenarios in our context) P of the size p_s (lines 1–3). Until the stopping criterion is met (e.g., a predefined computational time budget is exhausted), the algorithm repeats the following: (1) generating a new set of test cases Q using genetic operators from P using *crossover* and *mutation* (line 5), (2) generating another set of test cases W by merging P and Q and calculating their fitness scores by executing a simulator for every candidates in W (line 6), (3) updating A and U using W and E such that U excludes the objectives that are covered (achieved) by W for E and A includes the test cases from $A \cup W$, which are best at achieving the covered objectives (line 7), and (4) generating the next generation P from W considering U using *selection* (line 8). The algorithm ends by returning A (line 10).

The main differences between MOSA and FITEST are in the *initialPopulation* function (line 3) and the *generateNextGen* function (line 8): MOSA initializes P as a set of randomly generated test cases and keeps $|P| = p_s$ (typically p_s is set to $|O|$), whereas FITEST uses an adaptive random generation technique [19] to promote initial diversity and keeps reducing $|P|$ as $|U|$ decreases.

5.2.2 Surrogate-Assisted Many-Objective Search

Algorithm 2 appears to be suitable for solving the problem of test suite generation for DADS for online testing. However, iteratively evaluating the fitness scores of candidate test scenarios using simulations may take a prohibitive amount of time, preventing the generation of an effective test suite within a reasonable

time budget. To address this, we present a novel algorithm, called Surrogate-Assisted Many-Objective Test suite generation Algorithm (SAMOTA), that extends Algorithm 2 to effectively utilize surrogate models.

Following state-of-the-art surrogate-assisted optimization algorithms [134, 120, 78], SAMOTA uses the idea of iterating two search phases, namely *global* search and *local* search. Briefly speaking, global search (with global surrogate models) first *explores* the search space by capturing the global outline of the fitness landscape, and then local search (with local surrogate models) *exploits* the promising areas found by the global search. Only the best predicted test cases that are found through cooperation between global and local search are evaluated through expensive simulations, whose results will be used to build more accurate surrogate models in the next iteration, thus iteratively finding more promising test cases. This search process continues until the computational budget is exhausted. Since it is usually difficult to accurately approximate the whole search space relying on global surrogate models only, such cooperation between global and local search is more effective at achieving the search objectives [154]. The details of global and local searches will be provided below.

In addition to applying the state-of-the-art, surrogate-assisted optimization algorithms to the important software engineering problem of test suite generation, we address its limitations in our context. We observed that these algorithms generate either too many local surrogate models that tend to degrade performance or only one local surrogate model that cannot accurately capture the local fitness landscape of individual promising areas. To address the issues, we introduce a novel, clustering-based approach that generates one local surrogate model per cluster composed of test cases that belong to the same promising area. Before we move on to the details, we first provide an overview of SAMOTA below.

Similar to Algorithm 2, SAMOTA (whose pseudo-code is shown in Algorithm 3) takes as input a set of objectives O, a population size p_s, and a set of error thresholds E, plus the maximum numbers of iterations for global search g_{max} and local search l_{max} (as stopping criteria for global and local search), the percentage of test cases to be used for training local surrogate models η, the minimum number of test cases in a cluster c_m for local search, and a database D that keeps all test cases already evaluated by the simulator (if any); SAMOTA then returns an archive A as in Algorithm 2 and the database D updated during the execution. The updated database can be used as input for future execution.

The algorithm begins with initializing A, the set of uncovered objectives U, and the set of test cases P of size p_s (lines 1-3). For the initialization of P while promoting diversity, an adaptive random generation technique [19] is used. The fitness scores of the test cases in P are then computed by executing a simulator (line 4) and updates A and U using P and E such that U excludes the objectives that are covered (achieved) by P with respect to the given error thresholds E and A includes the test cases from $A \cup P$, which are best at achieving the covered objectives (line 5). The algorithm also updates D to include P since the simulator is executed for all test cases in P (line 6). Until the stopping criterion is met, the algorithm repeats the surrogate-assisted global search (lines 8–11) and the surrogate-assisted local search (lines 12–15). For the global search, the algorithm generates the set of test cases $\hat{T}_g$ that are expected to satisfy U with respect to E using algorithm **GS** (line 8), calculates the fitness scores of the test cases in $\hat{T}_g$ by executing the simulator to generate the set of test cases T_g that also contains their actual fitness scores (line 9), and updates A, U, and D using T_g, as done for P (lines 10-11). For the local search, the algorithm repeats the same procedures as for the global search, except that it generates the set of test cases $\hat{T}_l$, that are expected to better satisfy U than $\hat{T}_g$, with respect to E using algorithm **LS** (line 12).

Algorithm 3: SAMOTA

Input : Set of Objectives O
Population Size p_s
Set of Error Thresholds E
Max Iteration for Global Search g_{max}
Max Iteration for Local Search l_{max}
Percentage of Test Cases for Local Search η
Minimum Number of Test Cases in Cluster c_m
Database D

Output : Archive A
Updated Database D

1 Archive $A \leftarrow \emptyset$
2 Set of Uncovered Objectives $U \leftarrow O$
3 Set of Test Cases $P \leftarrow$ *InitialPopulation*(p_s)
4 $P \leftarrow$ *calculateFitnessSim*(P)
5 $A, U \leftarrow$ *updateArchive*(A, P, E, O)
6 $D \leftarrow$ *updateDatabase*(D, P)
7 **while** *not (stopping-condition)* **do**
8 Set of Test Cases $\hat{T}_g \leftarrow$ **GS**(D, U, p_s, g_{max}, E)
9 Set of Test Cases $T_g \leftarrow$ *calculateFitnessSim*$(\hat{T}_g)$
10 $A, U \leftarrow$ *updateArchive*(A, T_g, E, O)
11 $D \leftarrow$ *updateDatabase*(D, T_g)
12 Set of Test Cases $\hat{T}_l \leftarrow$ **LS**$(D, U, l_{max}, c_m, \eta)$
13 Set of Test Cases $T_l \leftarrow$ *calculateFitnessSim*$(\hat{T}_l)$
14 $A, U \leftarrow$ *updateArchive*(A, T_l, E, O)
15 $D \leftarrow$ *updateDatabase*(D, T_l)
16 **end**
17 **return** A, D

The algorithm ends by returning A and D (line 17). Note that $|\hat{T}_g|$ and $|\hat{T}_l|$ decrease as $|U|$ decreases, resulting in further reducing the number of expensive executions of the simulator.

GS (Global Search)

This algorithm aims to explore the search space using global surrogate models for uncovered objectives. It basically uses the same search framework as Algorithm 2 and returns the *best* test case for each uncovered objective in terms of the fitness score *predicted* by the global surrogate model trained using all the test cases in the database. Recall that the resulting test cases will be evaluated using the simulator (line 9 in Algorithm 3) to calculate their actual fitness scores, and the database will be updated to include the test cases with their actual fitness scores, leading to more accurate surrogate models in the next iteration of **GS**. To further improve the accuracy, **GS** additionally finds and returns the *most uncertain* test case for each uncovered objective based on the uncertainty of the surrogate model predictions, which can be measured, for example, according to the magnitude of the disagreement among the outputs of the members of an ensemble surrogate model as explained in section 2.3.

Specifically, the **GS** algorithm (Algorithm 4) takes as input the database D, the set of uncovered objectives U, the population size p_s, the maximum number of iterations g_{max}, and the set of error thresholds E; it returns a set of resulting test cases $\hat{T}_g$ found by global surrogate models trained using D.

$\hat{T}_g$ consists of the most promising test case for each uncovered objective $u \in U$ and the most uncertain test case for each u, leading to $|\hat{T}_g| \leq |U| \times 2$.

Algorithm 4: GS (Global Search)

Input : Database D
Set of Uncovered Objectives U
Population Size p_s
Max Iteration g_{max}
Set of Error Thresholds E
Output : Set of Test Cases $\hat{T}_g$

1 Set of Global Surrogates $M_g \leftarrow$ *trainGlobals*(D, U)
2 Set of Best Test Cases $\hat{T}_b \leftarrow \emptyset$
3 Set of Most Uncertain Test Cases $\hat{T}_n \leftarrow \emptyset$
4 Integer Counter $i \leftarrow 0$
5 Set of Test Cases $P \leftarrow$ *initialPopulation*(p_s)
6 **while** $i < g_{max}$ **do**
7 Set of Test Cases $Q \leftarrow$ *genOffspring*(P)
8 Set of Test Cases $W \leftarrow$ *calcFitnessGS*$(P \cup Q, M_g)$
9 $\hat{T}_b, \hat{T}_n, U \leftarrow$ *update*$(\hat{T}_b, \hat{T}_n, W, U, E)$
10 $P \leftarrow$ *generateNextGen*(W, U)
11 $i \leftarrow i + 1$
12 **end**
13 **return** $\hat{T}_b \cup \hat{T}_n$

The algorithm begins with training the set of global surrogate models M_g (one per an uncovered objective in U) using all the test cases in D (line 1). The algorithm then initializes the set of best test cases $\hat{T}_b$, the set of most uncertain test cases $\hat{T}_n$, the counter i, and the set of test cases P of size p_s (lines 2–5). While $i < g_{max}$, the algorithm repeats the following steps: (1) generate the offspring Q from P (line 7), (2) generate the set of test cases W by merging P and Q and predicting their fitness scores using M_g while recording the uncertainty of individual predictions (line 8), (3) update $\hat{T}_b$, $\hat{T}_n$, and U such that $\hat{T}_b$ includes the best test case from $\hat{T}_b \cup W$ for each $u \in U$, $\hat{T}_n$ includes the most uncertain test case from $\hat{T}_n \cup W$ for each $u \in U$, and U excludes the objectives covered by $\hat{T}_b$ (line 9), (4) generate the next generation P from W for U (line 10), and (5) increase i by 1 (line 11). The algorithm ends by returning $\hat{T}_g = \hat{T}_b \cup \hat{T}_n$ (line 13).

Note that each global surrogate model should be able to provide the uncertainty of individual predictions in addition to predicted fitness scores. An ensemble model already satisfies the requirement as the disagreement among the outputs of the ensemble members can be used to measure uncertainty (see chapter 2 for more details). Following the widely used surrogate models in the area of surrogate-assisted optimization [78, 134], we combine Kriging, polynomial regression, and radial basis function network models into an ensemble surrogate model for global search to accurately provide fitness score predictions and easily calculate the uncertainty of individual predictions.

LS (Local Search)

This algorithm aims to exploit promising areas, found by the global search, using local surrogate models. For each promising area for each uncovered objective, it finds and returns the best predicted test case based

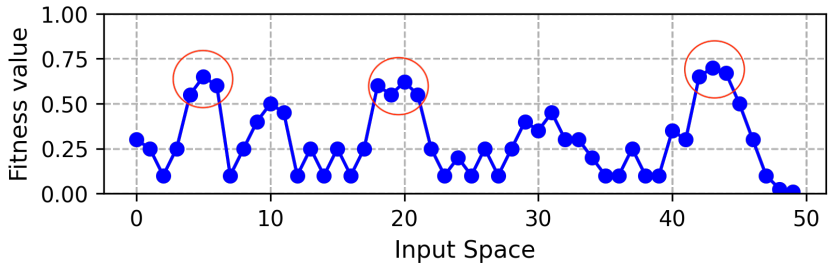


Figure 5.1: Illustration of clustering-based, local surrogate model generation

on a single-objective search. Since inexpensive surrogate models are used for fitness evaluations, we can use population-based optimization algorithms, such as Genetic Algorithm [138], rather than single-state optimization algorithms to increase the search performance. An important challenge is how to train a local surrogate model that accurately captures the local fitness landscape of a certain area.

An approach proposed by Zhou et al. [154] builds a local surrogate model using the m nearest data points in the database for each of the top $\eta\%$ individuals (in terms of their actual fitness scores) in the database. While each surrogate model can intuitively represent the local fitness landscape in the vicinity of a good individual, the number of the surrogate models can be an issue (especially considering the growth of the database) because the local search should be iterated for each individual and uncovered objective to find the best test case. Another approach proposed by Wang et al. [134] builds one local surrogate model for all top $\eta\%$ individuals at once. While this is clearly better than the former solution in terms of number of surrogate models, the top $\eta\%$ individuals can be too widespread, making the surrogate model unable to accurately capture the *local* fitness landscape.

To address the limitations of existing approaches in our context, we introduce a clustering-based approach for local surrogate model generation, which combines the benefits of the two approaches described above by limiting the number of surrogate models while avoiding the combination of top individuals that are too far away from each other. Our approach once again first selects the top $\eta\%$ individuals. But it then clusters the selected individuals based on their vicinity in the fitness landscape. Based on the clustering results, our approach builds one local surrogate model for each cluster. Taking Figure 5.1 as a simple uni-dimensional example where the dots denote all test cases in the database and the red circles indicate the clusters generated for the top 10 test cases. While the individuals are widespread in the fitness landscape, a local surrogate model is built based on the test cases in each cluster, to allow the local search to exploit the best candidates located in a specific area.

For clustering test cases, we use Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [85], with an optional parameter specifying the minimum number of data points in each cluster. One can select the minimum number if the minimum amount of training data is already known for a certain surrogate model type to be used; otherwise, the default value of 5 provided by HDBSCAN can be used.

Algorithm 5 presents the pseudo-code for local search, including our clustering-based local surrogate model generation. It takes as input the database D, the set of uncovered objectives U, the maximum number of iterations l_{max}, the percentage of test cases in D to be used for training local surrogate models η, and the minimum number of data points in a cluster c_m; it returns a set of test cases $\hat{T}_l$ that are expected to be the best at satisfying U according to local surrogate models. Note that $\hat{T}_l$ may contain multiple test

Algorithm 5: LS (Local Search)

Input : Database D
Set of Uncovered Objectives U
Max Iteration l_{max}
Percentage for Training Surrogate Models η
Minimum Number of Test Cases in Cluster c_m
Output : Set of Test Cases $\hat{T}_l$

```
1  Set of Test Cases T̂_l ← ∅
2  foreach Objective u ∈ U do
3      Set of Clusters C ← generateClusters(D, η, u, c_m)
4      foreach Cluster (Set of Test Cases) P ∈ C do
5          Surrogate Model m_u ← trainLocal(P)
6          Test Case T̂_b ← null
7          Integer Counter i ← 0
8          while i < l_max do
9              Set of Test Cases Q ← genOffspring(P)
10             Set of Test Cases W ← calcFitnessLS(P ∪ Q, m_u)
11             T̂_b ← updateBestPredicted(T̂_b, W)
12             P ← generateNextGen(W)
13             i ← i + 1
14         end
15         T̂_l ← T̂_l ∪ {T̂_b}
16     end
17 end
18 return T̂_l
```

cases for one objective if there are multiple promising areas for one objective.

Algorithm 5 begins with initializing a set of test cases $\hat{T}_l$ (line 1). For each uncovered objective $u \in U$, the algorithm finds the best test cases (possibly many if there are many promising areas) to be added into $\hat{T}_l$ (lines 2–15). Specifically, the algorithm generates clusters, with the minimum number of test cases c_m in each cluster, from the top $\eta\%$ test cases in D[1] for u (line 3), trains a local surrogate model for each cluster (lines 4–5), and finds the best predicted test case $\hat{T}_b$ using the local surrogate model (lines 6–13). The algorithm ends by returning $\hat{T}_l$ (line 18).

Unlike **GS**, **LS** does not use the uncertainty of surrogate models' predictions. Therefore, considering computationally efficiency, we can use any of the non-ensemble models described in section 2.3. We will show how to find the best configuration for **LS**, including the surrogate model type, in our empirical evaluation (see section 5.3.2).

5.3 Empirical Evaluation

This section reports the empirical evaluation of our approach for efficient DNN testing when applied to an open-source DADS. Specifically, we investigate the following research questions:

RQ1: What is the best configuration for `LS`?

[1] If the number of the top $\eta\%$ test cases in D is less than c_m, then the top c_m test cases in D are used.

RQ2: How do alternative approaches fare in terms of test effectiveness?

RQ3: How do alternative approaches fare in terms of test efficiency?

RQ1 aims to find the best configuration for LS before we investigate the effectiveness and efficiency of SAMOTA. We propose a new clustering-based approach for better local surrogate model generation in LS. However, compared to existing approaches, we need to assess our clustering-based approach in terms of the effectiveness of LS. Furthermore, its impact may vary depending on the types of surrogate models (e.g., Kriging, polynomial regression, and radial basis function network). To answer these questions, we compare the combinations of surrogate model generation approaches and surrogate model types in terms of the ability of LS to find the most critical test inputs for a given time budget. Notice that we do not investigate the best configuration for GS since this has already been investigated in existing studies [76, 78] and we will therefore rely on reported results.

Using the best configuration for LS resulting from answering RQ1, RQ2 and RQ3 aim to investigate the effectiveness and efficiency of SAMOTA, respectively, in comparison to "naive" approaches that do not use surrogate models, such as MOSA, FITEST, and Random Search (with archive). To answer RQ2, we investigate how many safety violations are found by test suites generated using different approaches given a time budget. To answer RQ3, we investigate how quickly target safety violations are found by test suites generated using different approaches. The answers will show how effective and efficient SAMOTA can be by adapting the idea of surrogate-assisted optimization.

We conducted our evaluation on Ubuntu 18.04 running on Intel i9-9900K CPU with RTX 2080 Ti (11 GB) and 32 GB memory.

5.3.1 Case Study Subjects

We use Pylot [46], a publicly available DADS, as our case study subject. To enable simulation-based testing, we also use CARLA [30], a high-fidelity, open-source simulator for ADS.

Pylot is a DADS for developing and testing autonomous vehicle components (e.g., perception, prediction, planning) on the CARLA simulator and real-world vehicles [46]. Given a driving environment (either simulated or real), it drives the ego vehicle by controlling its acceleration, braking, and steering according to input data dynamically collected though sensors (e.g., camera and LiDAR). To achieve this, it consists of multiple components providing various functions of an autonomous vehicle, such as traffic light detection, lane detection, and object tracking. For each component, Pylot provides the implementations of state-of-the-art approaches based on pre-trained DNNs. For example, SSD (Single Shot Detector) [79] is used for object detection while SORT (Simple Online and Realtime Tracking) [13] and DeepSORT [142] enable obstacle tracking.

CARLA [30] is an open-source simulator based on the Unreal Engine [34], designed to support training, development, and validation of ADS. CARLA provides hand-crafted, high-fidelity virtual maps having various static environments, such as different road types (e.g., straights and curves), different sizes of buildings, different shapes of trees, and different positions for traffic lights. Such configurable attributes can be used to define the test input space of DADS. In our evaluation, we consider as many configurable attributes as possible: road type, start/end-point on maps, the presence of other vehicles in front/same/opposite lane, other vehicle types, vehicle speed, the density of pedestrians, the presence of trees and buildings, time of day, and weather condition. To avoid invalid scenario generation (e.g., the

road type is 'straight' while the start-point on a map is 'at the start of a curve road'), we use additional pre-defined constraints on the attribute values. More details are provided in the supporting materials [131].

Considering the capability of CARLA to compute related metrics (e.g., the distance between vehicles), we use the following six (safety) requirements for Pylot: (1) follow the center of the lane, (2) avoid collision with other vehicles, (3) avoid collision with pedestrians, (4) avoid collision with static objects (e.g., traffic signs), (5) abide by traffic rules (e.g., traffic lights), and (6) reach the destination within a given time. More details about the requirements and their implementations are provided in the supporting materials.

We selected the combination of Pylot and CARLA as our case study subject since (1) Pylot is an advanced, DNN-based ADS composed of multiple autonomous driving components, (2) Pylot is designed to be easily usable in simulated environments created by CARLA, and (3) both are publicly available. While Apollo[2] is another DADS one could rely on in our investigation, we do not use it because of compatibility issues (e.g., not compatible with the latest version of CARLA) and incomplete implementations (e.g., its camera perception module is not available).

5.3.2 RQ1: Best Configuration for Local Search

Setup

To answer RQ1, we generate a set of test cases by executing **LS** using different configurations and measure the *Local Search Effectiveness* (*LSE*) of the test set. Since **LS** aims to return a set of test cases that are expected to be the best at satisfying the given search objectives (i.e., violating safety requirements) by *predicting* the fitness scores of the test cases, the *LSE* of a test set can be measured by its *actual* fitness scores for all objectives. Specifically, we measure the *LSE* of a set of test cases T as $LSE(T) = \frac{\sum_{o \in O} \max_{t \in T} f(t,o)}{|O|}$ where O is a set of objectives and $f(t, o)$ is the (normalized) actual fitness score of a test case $t \in T$ for an objective $o \in O$. In other words, we calculate the maximum actual fitness score achieved when running all test cases in T for each objective in O and average these scores across all objectives.

Regarding the **LS** configurations, we consider the combinations of surrogate model types and generation approaches. Surrogate model types include KriGing (KG) [118], Polynomial Regression (PR) [119], and Radial basis Function network (RF) [16] as they are the most widely used in the literature [78, 45, 134, 11, 91, 39, 42]. For RF, we set the number of neurons in the hidden layer to 10 since it gave the best accuracy at predicting the fitness scores of test cases in our preliminary evaluation. Similarly, we set the degree of polynomial models for PR to 2 based on our preliminary evaluation. Regarding surrogate model generation, there are three different approaches for a set of data points (i.e., test cases with actual fitness scores): (1) build one surrogate model using *all* the data points; (2) build one surrogate model for each data point and its *neighbors*; and (3) build one surrogate model for each cluster after *clustering* the data points. The first and second approaches (i.e., *all* and *neighbors*) come from existing work [154, 134] while the third approach is newly proposed in this chapter. However, we decided to exclude the second approach because it generates too many surrogate models to process as the number of data points provided to **LS** increases during the execution of SAMOTA. As a result, we consider a total six configurations (i.e., the combination of 3 surrogate model types and 2 surrogate model generation

[2]https://github.com/ApolloAuto/apollo

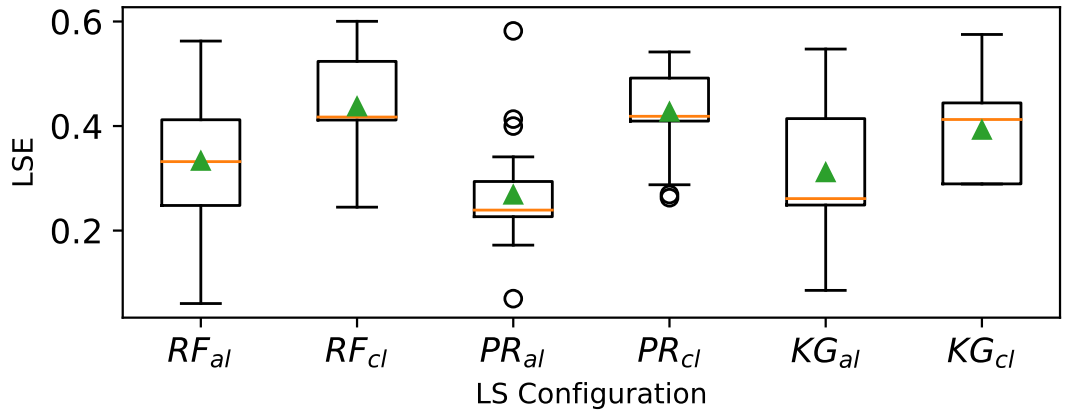


Figure 5.2: Distribution of *LSE* values for different **LS** configurations

approaches). For simplicity, an **LS** configuration is denoted by X_Y where $X \in \{RF, PR, KG\}$ refers to the surrogate model type and $Y \in \{al, cl\}$ refers to the surrogate model generation approach (i.e., *all* and *clustering*). For example, KG_{cl} denotes the configuration that uses Kriging and the *clustering* approach for model generation.

To run **LS**, we need to set its inputs: the database D, the set of target objectives U, the maximum number of iterations l_{max}, the percentage of data points in D to be used for training local surrogate models η, and the minimum number of data points in a cluster c_m. To generate diverse test scenarios in D, we use 4-way combinatorial coverage for all the attributes used to define the test input space, resulting in $|D| = 587$. We select $l_{max} = 200$, based on our preliminary evaluations, since increasing l_{max} above 200 no longer significantly increases the predicted fitness scores of the test cases generated by **LS**. We select $\eta = 20\%$ following the recommendations of a recent paper [78]. For c_m, we use the default value of 5 provided by HDBSCAN [85].

To account for randomness in **LS**, we repeat the experiment 20 times. We apply the non-parametric Mann–Whitney U test [83] to assess the statistical significance of differences in *LSE* across **LS** configurations. Since we statistically test five hypotheses for each **LS** configuration (as we compare the six **LS** configurations pairwise), we use a level of significance $\alpha = 0.05/5 = 0.01$ by applying the Bonferroni correction [137] to reduce the risk of Type 1 errors. We also measure Vargha and Delaney's $\hat{A}_{AB}$ [132] to capture the effect size of the difference, which can be typically characterized as small, medium, and large when the $\hat{A}_{AB}$ value exceeds 0.56, 0.64, and 0.71, respectively. Note that $\hat{A}_{AB} = 1 - \hat{A}_{BA}$ and $\hat{A}_{AB} = \hat{A}_{BA} = 0.5$ means there is no statistical difference between the two compared configurations.

Results

Figure 5.2 shows the distribution of the *LSE* values for the six **LS** configurations. The orange bar and the green triangle in the middle of each box represent the median and the average, respectively. Table 5.1 additionally shows the results of statistical comparisons between different **LS** configurations. The columns *A* and *B* indicate the two configurations being compared. The columns *p-value* and $\hat{A}_{AB}$ indicate the statistical significance and effect size, respectively, when comparing *A* and *B* in terms of *LSE*.

Let us first compare the two surrogate model generation approaches, i.e., al and cl, for the same surrogate model type. In Figure 5.2, for all surrogate model types, the cl approach seems better than the al approach. In Table 5.1, given a level of significance $\alpha = 0.01$, we can see that the difference between

Table 5.1: Statistical comparison results for **LS** configurations

A	B	p-value	$\hat{A}_{AB}$	A	B	p-value	$\hat{A}_{AB}$
RF_{cl}	RF_{al}	0.001	0.78	RF_{cl}	KG_{cl}	0.162	0.59
RF_{al}	PR_{al}	0.015	0.70	PR_{cl}	PR_{al}	0.000	0.90
PR_{cl}	RF_{al}	0.001	0.78	KG_{al}	PR_{al}	0.033	0.67
RF_{al}	KG_{al}	0.347	0.54	KG_{cl}	PR_{al}	0.000	0.86
KG_{cl}	RF_{al}	0.019	0.69	PR_{cl}	KG_{al}	0.001	0.79
RF_{cl}	PR_{al}	0.000	0.90	PR_{cl}	KG_{cl}	0.308	0.55
RF_{cl}	PR_{cl}	0.495	0.50	KG_{cl}	KG_{al}	0.003	0.76
RF_{cl}	KG_{al}	0.001	0.78	-	-	-	-

cl and al for the same surrogate model type is statistically significant in all cases. Furthermore, the $\hat{A}_{AB}$ value is always greater than 0.71, meaning that cl is largely better than al in terms of *LSE*. In particular, the difference between PR_{cl} and PR_{al} is extreme (p-value $= 0.000$ and $\hat{A}_{AB} = 0.90$). This is because the al approach yields outliers by considering all data points at once, thus making the surrogate models (especially PR) inaccurate, whereas the cl approach does not thanks to clustering. As a result, for the same surrogate model type, using the cl approach is clearly better, showing the practical usefulness of our clustering approach in local surrogate model generation.

The ranking of the surrogate model types, based on their average *LSE* values over 20 runs, is RF_{cl}, PR_{cl}, and KG_{cl}, respectively. However, with $\alpha = 0.01$, the differences between them are all insignificant, meaning that it does not make a difference, in terms of *LSE*, whether RF_{cl}, PR_{cl}, or KG_{cl} is used. Nevertheless, the results do not imply that there is no significant difference among RF, PR, and KG for all problems. If possible, in practice, it is better for engineers to determine the best surrogate model type for **LS** before running SAMOTA. While it requires a dataset D containing diverse test cases with actual fitness scores, such a dataset could be available if the system under test already went through system testing and the test results regarding safety requirements were recorded. Otherwise, one can opt for RF_{cl} as it is known to provide both computational efficiency and reasonable training accuracy [78]. In our evaluation, we therefore use RF_{cl} for the remaining RQs.

The answer to RQ1 is that our clustering-based approach (cl) for surrogate model generation is significantly better than the existing approach (al) in all cases but there is no practical difference overall between different surrogate model types. In practice, it is therefore better to experimentally determine the best local surrogate model type for a given system under test, while relying on cl for surrogate model generation. Otherwise, RF_{cl} can be the default option when this is not possible.

5.3.3 RQ2: Test Effectiveness

Setup

To answer RQ2, we generate a test suite using SAMOTA and its alternatives (e.g., MOSA and FITEST) for a fixed time budget and measure the *Test Effectiveness* (*TE*) of the test suite, defined as the proportion of safety requirements that are violated when running the test suite over the total number of safety requirements. *TE* ranges between 0 and 1, where higher values are desirable.

For SAMOTA, if we have a database that keeps test cases and their actual fitness scores computed in

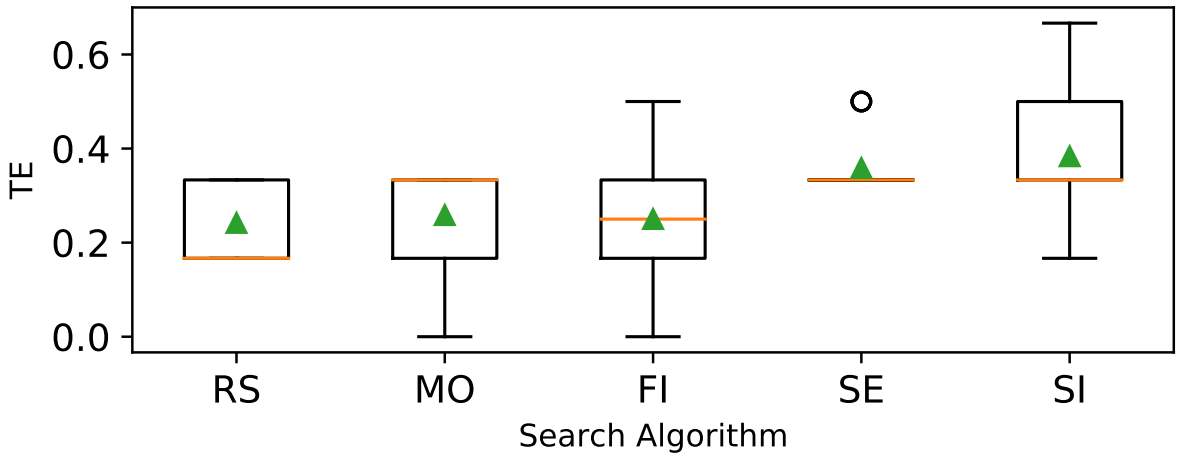


Figure 5.3: Distribution of *TE* values for different search approaches

previous testing sessions, the database can be provided as input (see Algorithm 3). Though SAMOTA can start without it, providing a non-empty database as an initial input can boost the effectiveness of SAMOTA by improving the accuracy of surrogate models early. To better understand this, we use two configurations, i.e., SAMOTA starting with an empty database (SAMOTA-E) and SAMOTA starting with an initial database (SAMOTA-I). For the initial database, we use 4-way combinatorial coverage based on all the attributes used to define the test input space to generate diverse test cases, as we did for RQ1.

As alternatives to SAMOTA, we use MOSA [96] and FITEST [4], as they are the state-of-the-art many-objective test suite generation algorithms. We also use Random Search (RS) that randomly generates test cases for each iteration, as a baseline. Similar to MOSA and FITEST, we use an archive in RS so that it keeps the best at satisfying individual objectives in the archive until the search ends; the resulting archive is a test suite generated by RS. RS will provide insight into how easy the search problem is and will help us evaluate the impact of using advanced search algorithms, such as MOSA, FITEST, and SAMOTA, on test effectiveness.

The initial population size of MOSA, FITEST, and SAMOTA is the number of objectives. To be consistent in terms of population size, we set the number of newly generated test cases at each iteration of RS to be the number of objectives. For the other parameters, such as mutation and crossover rates in MOSA, FITEST, and SAMOTA-E/I, we adapt the default values used by Fraser and Arcuri [38].

To account for randomness in all approaches, we repeat the experiment 20 times. For each run, we use the same budget of two hours, as we found that it was long enough to converge in our preliminary evaluation. We apply the Mann–Whitney U test [83] to assess the statistical significance of differences in *TE* among approaches. Since we statistically test four hypotheses for each approach (as we compare RS, MOSA, FITESET, SAMOTA-E, and SAMOTA-I pairwise), we use a level of significance $\alpha = 0.05/4 = 0.0125$ by applying the Bonferroni correction [137] as we did for RQ1. We also measure Vargha and Delaney's $\hat{A}_{AB}$ [132] to capture the effect size of the difference.

Results

Figure 5.3 shows the distribution of *TE* values achieved by RS, MOSA (MO), FITEST (FI), SAMOTA-E (SE), and SAMOTA-I (SI) over 20 runs. Again, the orange bar and the green triangle in the middle of each box represent the median and average, respectively. Table 5.2, whose format is the same as Table 5.1,

Table 5.2: Statistical comparison results for different search approaches

A	B	p-value	$\hat{A}_{AB}$	A	B	p-value	$\hat{A}_{AB}$
MO	FI	0.377	0.53	SI	MO	0.000	0.77
FI	RS	0.411	0.52	SE	MO	0.000	0.75
SI	FI	0.001	0.76	SI	RS	0.000	0.82
SE	FI	0.002	0.74	SE	RS	0.000	0.81
MO	RS	0.229	0.56	SI	SE	0.210	0.56

presents the results of statistical comparisons between different approaches. Notice that some safety requirements may never be violated, and therefore absolute *TE* values cannot be interpreted; we use these values only for comparison purposes.

Overall, the results show that SAMOTA-I is the best in terms of the average *TE* value for 20 runs. With a level of significance $\alpha = 0.0125$, the difference between SAMOTA-I and SAMOTA-E is insignificant (p-value $= 0.210$), but the differences between SAMOTA-I/E and the others are all significant with large effect sizes. This means that, by leveraging surrogate models, SAMOTA can be significantly more effective than the state-of-the-art test suite generation approaches and random search in terms of revealing unknown safety violations within a reasonable time budget.

Interestingly, the differences between MOSA and RS and between FITSET and RS are insignificant (p-values are 0.229 and 0.411, respectively), meaning that MOSA and FITEST are *not* significantly better than RS in terms of *TE*. A detailed analysis of the results shows that this is because the two-hour time budget is not enough for MOSA and FITEST to evaluate and evolve candidate test cases many times; on average, across 20 runs, only around five generations were completed during each run of MOSA and FITEST. In contrast, SAMOTA went through more than 800 generations using global and local surrogate models, within the same time budget, and therefore yielded significantly higher *TE* values than MOSA and FITEST as a result. This also confirms that the surrogate models of SAMOTA are sufficiently accurate to effectively guide the search towards test cases that cause safety violations. However, we would expect the difference between SAMOTA and MOSA or FITEST to diminish with a much longer time budget allowing them to go through many more generations. Nevertheless, such a scenario is unrealistic in practice as SAMOTA is likely to remain significantly more effective than its alternatives for practical time budgets.

Another interesting result is that there is no statistical difference between SAMOTA-I and SAMOTA-E, meaning that providing an initial database in SAMOTA does not lead to a significant improvement in detecting safety violations. This implies that SAMOTA can generate good enough test suites even without an initial database, an importance practical consideration. In RQ3, we will further compare SAMOTA-I and SAMOTA-E in terms of test efficiency (i.e., how fast safety violations are detected).

The answer to RQ2 is that, in our context, SAMOTA is significantly more effective than other many-objective search algorithms tailored for test suite generation. Furthermore, SAMOTA can achieve acceptable test effectiveness without an initial database.

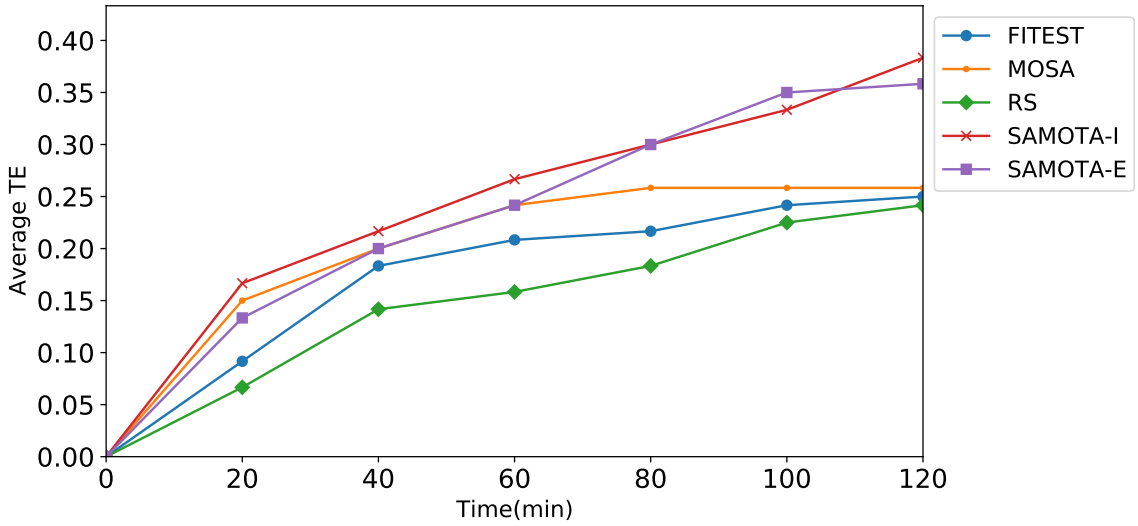


Figure 5.4: Test efficiency for different search approaches

5.3.4 RQ3: Test Efficiency

Setup

To answer RQ3, we use the same approaches and setups as in RQ2 (i.e., SAMOTA-I/E, MOSA, FITEST, and RS). We generate a test suite using each approach and measure its *execution time* to achieve specific *TE* values (i.e., 1/6, 2/6, ..., 6/6 as there are six safety requirements in total).

To account for randomness, as we did in RQ2, we repeat the experiment 20 times. Notice that we cannot calculate the average execution time for 20 runs to achieve a specific *TE* value because not all 20 runs necessarily achieve such *TE* value (even for *TE* = 1/6). Therefore, we compute how the average *TE* values for 20 runs vary over time from 20*min* to 120*min*, in steps of 20*min*.

Results

Figure 5.4 shows the relationship between the execution time and the average *TE* values for 20 runs across all approaches. For example, RS is always at the bottom, meaning that, on average, RS achieves the lowest *TE* values compared to the others over the same time period.

Comparing SAMOTA-I and SAMOTA-E, we can see that SAMOTA-I achieves higher *TE* values than SAMOTA-E for the first 60*min* on average, but this difference vanishes after 80*min*. This is because the surrogate models of SAMOTA-E are relatively inaccurate in the beginning, as no initial database was provided, but they get more accurate over time as the database grows. Such growth also explains why MOSA achieves a higher average *TE* value than SAMOTA-E after 20*min* and why this trend is reversed after 80*min*.

Comparing SAMOTA-I/E and alternatives, SAMOTA-I is always at the top, meaning that it is always faster than the alternatives to achieve the same level of test effectiveness. This is the same for SAMOTA-E, except for the first 40*min* where it is slower than MOSA for the reason provided above.

The answer to RQ3 is that SAMOTA is more efficient than alternative test suite generation approaches using many-objective search as soon as its surrogate models become sufficiently accurate. An initial database can boost the efficiency of SAMOTA in the initial search phase and allow it to surpass other techniques right from the start.

5.3.5 Threats to Validity

Since we use a specific DES (i.e., Pylot), coupled with a simulator (i.e., CARLA), external validity is our main challenge here. However, Pylot and CARLA are respectively representative of advanced DADS and high-fidelity driving simulators, in terms of accuracy, fidelity, and performance [46, 30]. Furthermore, no other realistic DADS, coupled with a high-fidelity simulator, is publicly available at this point, which is indeed an impediment to further experiments on this topic. Note that such experiments would likely be highly computationally-intensive, given that it took more than 300 computing hours in our case. Nevertheless, further experiments with different DES and high-fidelity simulators would be required to strengthen the generalizability of our results.

We want to remark that the applicability of SAMOTA is not limited to a specific DES since none of the algorithms (3-5) assumes specific DES properties. As long as there are many safety requirements (possibly independent from each other) for a DES under test and the fitness evaluation for each requirement is expensive, SAMOTA would show promising results as compared to the other algorithms that do not use surrogate models.

5.4 Related work

5.4.1 Online Testing for DNN-Enabled Systems

In recent years, online testing for DNN-Enabled Systems (DES) has attracted more attention, especially in the context of ADS.

Gambi et al. [41] presented ASFAULT, a tool for automatically generating road networks based on a genetic algorithm to test if the ego vehicle under test keeps the center of the lane while driving in a simulated environment. Majumdar et al. [82] presented PARACOSM, a language and tool to systematically define and generate test scenarios for autonomous driving simulations. They used a fuzzing-based test input generation strategy to achieve high combinatorial coverage for the attribute values that define the test input space. Tuncali et al. [127] presented SIM-ATAV, a testing framework to generates test cases, using covering arrays and requirements falsification methods, for autonomous vehicle with machine learning components. Seymour et al. [111] presents an empirical study, in which they generated test cases, using metamorphic and equivalent partitioning techniques, to test DADS in black-box settings. Riccio and Tonella [104] presented DEEPJANUS, a search-based approach to generate similar input pairs, which causes the DADS under test, with a focus on lane keeping, to mispredict for one input and work fine for the other input. Despite this significant body of work, no existing study focuses on performing efficient online testing for DES, with many objectives (requirements), using surrogate models.

One notable exception is the study of Abdessalem et al. [3] as they used ML models (i.e., decision trees) to better guide the search process towards promising areas for efficient online testing for vision-based DADS. Nevertheless, their approach is inherently different from ours since their models are not

surrogate models that can replace the computationally expensive fitness evaluations with much less expensive approximations but classification models that help focus the search on the critical test input space. Furthermore, they used the classification models outside the search algorithm (i.e., NSGA-II) to reduce the search space, whereas we use surrogate models inside the search algorithm to calculate fitness scores without using expensive simulators. In fact, their approach is based on learnable evolutionary search [88], whereas our approach is based on surrogate-assisted optimization [61]. Therefore, the two approaches are orthogonal, meaning that one can easily combine the two approaches together; for example, one can use SAMOTA instead of NSGA-II for the approach of Abdessalem et al. [3] and iteratively reduce the search space using decision trees while reducing the execution time of fitness evaluations using surrogate models in the main search algorithm.

5.4.2 Surrogate-Assisted Optimization

Addressing computationally expensive optimization problems using surrogate models (see section 2.3 for details) has been widely studied in the field of surrogate-assisted optimization [61].

Among many studies, Zhou et al. [154] presented a pioneering idea of combining global and local surrogate models to accelerate evolutionary optimization. Specifically, they used global surrogate models to filter some promising individuals, and utilized local surrogate models that represent the local fitness landscape in the vicinity of the individual to accelerate convergence. The experimental results on multimodal benchmark functions and a real-world aerodynamic shape design problem showed that the idea of combining global and local searches yields significant savings in computational cost when compared to alternatives.

Following up on the idea of combining global and local searches, Wang et al. [134] additionally used the prediction uncertainty of global surrogate models to select candidates for actual fitness evaluations. By providing the actual fitness scores of the most uncertain candidates, it maximizes the information gain of the surrogate models, making them more accurate faster. Recently, Liu et al. [78] also experimentally confirmed that using the most uncertain candidates in addition to the best predicted candidates was a promising strategy for solving benchmark problems with up to 30 dimensions.

However, no existing study addresses surrogate-assisted optimization, combining global and local surrogate models, applied to the problem of test suite generation for DES online testing.

5.5 Conclusion

In this chapter, we present SAMOTA, a novel approach to effectively and efficiently generate test data for DNN-enabled systems in the context of online testing. In essence, it provides a strategy to effectively combine surrogate-assisted optimization and many-objective search. Empirical evaluation results on an advanced DNN-enabled ADS, with a high-fidelity driving simulator, show that SAMOTA is significantly more effective and efficient, with a large effect size, than the state-of-the-art many-objective test suite generation algorithms and random search.

5.6 Data Availability

The replication package of our experiments — including the implementation of search algorithms, simulator, and the details of the experimental setup — is available on Figshare [131].

Chapter 6

Many-Objective Reinforcement Learning for Test Suite Generation

In this chapter, we present MORLOT (Many-Objective Reinforcement Learning for Online Testing), a novel online testing approach for DNN-enabled systems (DES). MORLOT leverages two distinct approaches: (1) Reinforcement Learning (RL) [121] to generate the sequences of changes to the dynamic elements of the environment with the aim of causing requirements violations, and (2) many-objective search [96, 4] to efficiently satisfy as many independent requirements as possible within a limited testing budget.

Existing online testing approaches for DES exhibit at least one of two critical limitations. First, they do not account for the fact that there are often many safety and functional requirements, possibly independent of each other, that must be considered together in practice. Though one could simply repeat an existing test approach for individual requirements, it is inefficient due to its inability to dynamically distribute the test budget (e.g., time) over many requirements according to the feasibility of requirements violations. For example, if one of the requirements cannot be violated, the pre-assigned budget for this requirement would simply be wasted. Furthermore, dividing the limited test budget across many requirements may result in too small a budget for testing individual requirements thoroughly. Second, they do not vary dynamic environmental elements, such as neighboring vehicles and weather conditions, during test case (i.e., test scenario) execution. For example, certain weather conditions (e.g., sunny) remain the same throughout a test scenario, whereas in reality they may change over time, which can trigger requirements violations. This is mainly because the number of possible test scenarios increases exponentially when considering individual dynamic elements' changes (i.e., time series). However, not accounting for such dynamic environments could significantly limit test effectiveness by limiting the scope of the test scenarios being considered.

The combination of RL and many objective search works as follows: (1) RL incrementally generates the sequence of changes for the dynamic elements of the environment, and (2) those changes are determined by many-objective search such that they are more likely to achieve any of the uncovered objective

(i.e., the requirement not yet violated). In other words, changes in the dynamic elements tend to be driven by the objectives closest to being satisfied. For example, if there are three objectives o_1, o_2, and o_3 where o_2 is closest to being satisfied, MORLOT incrementally appends those changes that help in achieving o_2 to the sequence. Furthermore, by keeping a record of uncovered objectives as the state-of-the-art many-objective search for test suite generation does, MORLOT focuses on the uncovered ones over the search process.

Though MORLOT can be applied to any DES interacting with an environment including dynamically changeable elements, it is evaluated on DNN-enabled Autonomous Diving Systems (DADS). Specifically, we use Transfuser [102], the highest ranked DADS, at the time of our evaluation, among publicly available ones in the CARLA Autonomous Driving Leaderboard [124] and CARLA [30], a high-fidelity simulator that has been widely used for training and validating DADS. Our evaluation results, involving more than 600 computing hours, show that MORLOT is significantly more effective and efficient at finding safety and functional violations than state-of-the-art, many-objective search-based testing approaches tailored for test suite generation [96, 4] and random search.

Our contributions can be summarized as follows:

- MORLOT, a novel approach that efficiently generates complex test scenarios, including sequential changes to the dynamic elements of the environment of the DES under test, by leveraging both RL and many-objective search;

- An empirical evaluation of MORLOT in terms of test effectiveness and efficiency and comparison with alternatives;

- A publicly available replication package, including the implementation of MORLOT and instructions to set up our case study.

Significance. For DES that continuously interact with their operational environments and have many safety and functional requirements to be tested, performing online testing efficiently to identify as many requirements violations as possible, without arbitrarily limiting the space of test scenarios, is essential. However, taking into account the sequential changes of dynamic elements of an environment over time is extremely challenging since it renders the test scenario space exponentially larger as test scenario time increases. MORLOT addresses the problem by leveraging and carefully combining RL and many-objective search, thus providing an important and novel contribution towards scalable online testing of real-world DES in practice.

Chapter Structure. The rest of the chapter is structured as follows. Section 6.1 formalizes the problem of DES online testing with a dynamically changing environment. Section 6.2 describes our proposed approach, starting from a generic RL-based test generation approach and then MORLOT, a many-objective reinforcement learning approach for online testing. Section 6.3 evaluates the test effectiveness and efficiency of MORLOT using an open-source DNN-based ADS with a high-fidelity driving simulator. Section 6.4 discusses and contrasts related work. Section 6.5 concludes the chapter. Section 6.6 provides details on the replication package.

6.1 Problem Description

In this section, we provide a precise problem description regarding the automated online testing of DNN-enabled systems (DES) by dynamically changing their environment during simulation. As a working example, we use a DNN-enabled autonomous driving system (DADS) to illustrate our main points, but the description can be generalized to any DES.

In online testing, a DADS under test is embedded into and interacts with its driving environment. However, because of the risks and costs it entails, online testing is usually performed with a simulator rather than on-road vehicle testing. Using a simulator enables the control of the driving environment, such as weather conditions, lighting conditions, and the behavior of other actors (e.g., vehicles on the road and pedestrians). During simulation, the DADS continuously interacts with the environment by observing the environment via the ego vehicle's sensors (e.g., camera and LIDAR) and driving the ego vehicle through commands (e.g., steering, throttle, and braking). Due to the closed-loop interaction between the DADS and its environment, simulation is an effective instrument to check if any requirements violation can occur under realistic conditions. Notice that such violations can be triggered by dynamically changing the driving environment during simulation; for example, certain changes to the speed of the vehicle in front, which can often occur in practice due to impaired driving or sudden stops, can cause a collision. The goal of DADS online testing, based on dynamically changing the environment, is to find a minimal set of test cases, each of them changing the environment in a different way, to cause the DADS to violate as many requirements as possible.

Specifically, let d be the DADS under test on board the ego vehicle and $E = (X, K)$ be the environment where $X = \{x_1, x_2, \dots\}$ is a set of *dynamic* elements of the simulation (e.g., actors other than the ego vehicle, and the environment conditions) and K is a set of *static* elements (e.g., roads, buildings, and trees). Each dynamic element $x \in X$ can be further decomposed into a sequence $x = \langle x^1, x^2, \dots, x^J \rangle$ where J is the duration of the simulation and x^j is the value of x at time $j \in \{1, 2, \dots, J\}$ (e.g., the position, speed, and acceleration of the vehicle in front at time j). Based on that, we can define $X^j = \{x^j \mid x \in X\}$ as capturing the set of values of all the dynamic elements in X at time j and $E^j = (X^j, K)$ as indicating the set of values of all environmental elements (both dynamic and static) in E at time j. Let a *test case* $t = \langle X^1, \dots, X^J \rangle$ be a sequence of values of the dynamic elements for J time steps. By running d in E with t, using a simulator, at each time step $j \in \{1, 2, \dots, J\}$, d observes the snapshot of $E^j = (X^j, K)$ using the sensors (e.g., camera and LIDAR) and generates driving commands C^j (e.g., throttle, steering, and braking) for the ego vehicle. Note that what d observes from the same E varies depending on the ego vehicle's dynamics (e.g., position, speed, and acceleration) computed by C; for example, d observes the *relative* distance between the ego vehicle and the vehicle in front, which naturally varies depending on the position of the ego vehicle. For the next time step $j + 1$, the simulator computes $E^{j+1} = (X^{j+1}, K)$ according to t and generates what d observes from E^{j+1} by taking into account C^j.

Given a set of requirements (safety, functional) R, the degree of a violation for a requirement $r \in R$ produced by d in E for t at any time step $j \in \{1, \dots, J\}$, denoted by $v(r, d, E, t, j)$, can be measured by monitoring the simulation of d in E for t. For example, the distance between the ego vehicle and the vehicle in front can be used to measure how close they are from colliding. If $v(r, d, E, t, j)$ is greater than a certain threshold ϵ_r (i.e., the distance is closer than the minimum safe distance) at any j, we say that d *violates* r at j. Let $v_{max}(r, d, E, t)$ be the maximum degree of violation for r produced by d in

E for t during simulation. Given an initial environment $E^1 = (X^1, K)$, the problem of DADS online testing for dynamically changing environments is to find a minimal set of test cases TS that satisfies $v_{max}(r, d, E, t) > \epsilon_r$ for as many $r \in R$ as possible when executing all $t \in TS$.

Test data generation for online testing of a DADS, with a dynamically changing environment, presents several challenges. First, the input space for dynamically changing the behavior of the environment is enormous because there are many possible combinations of environmental changes for each timestamp. Second, there are usually many independent requirements to be considered simultaneously. For example, keeping a safe distance from the vehicle in front is independent from the ego vehicle abiding by the traffic lights. If the requirements are not considered simultaneously, no practical test budget may be sufficient to thoroughly test each requirement, as a limited budget must be divided across all individual requirements. Third, in addition to the second challenge, depending upon the accuracy of the DADS under test, it may be infeasible to violate some requirements. This implies that the pre-assigned budgets for those requirements are inevitably wasted if a testing approach cannot simultaneously consider all requirements. Last but not least, a DADS is often developed by a third party, and as a result its internal information (e.g., about DNN models) is often not fully accessible. Therefore, online DADS testing must often be carried out in a black-box manner.

To address the challenges mentioned above, we propose a novel approach that combines two distinct approaches: (1) RL to dynamically change the environment based on the simulation state (including the state of the DES under test and the state of the environment) at each timestamp with the aim of causing requirements violations, and (2) many-objective search to effectively and efficiently achieve many independent objectives (i.e., violating requirements in our context). Furthermore, our approach is DNN-agnostic; it does not need any internal information about the DNN.

6.2 Reinforcement Learning-Based Test Generation

This section presents MORLOT (Many-Objective Reinforcement Learning for Online Testing), our novel approach to address the problem explained in Section 6.1. In the following subsections, we first describe how Reinforcement Learning (RL) can be tailored for the generation of a single test case (i.e., a test scenario in the context of DES online testing), and then present MORLOT by extending it.

6.2.1 Test Case Generation using RL

RL has widely been used to learn the sequence for completing a sequential decision-making task [121, 114]. RL has been also applied to automated software testing [150, 151, 94]. For the latter, RL is particularly suitable for systems whose usage entails sequential steps, for example ordering something from the web [150] such as: (1) going to the website, (2) putting something in the cart, (3) checkout and payment. Similarly, in the case of testing a DADS, we require sequential changes in the environment; for example, sequential steps for one scenario can be: (1) change the weather to *Rainy* (to decrease the friction between tyres and the road), (2) increase the *fog* level (to reduce visibility), (3) increase the speed of the vehicle-in-front (to increase distance from the ego-vehicle, which then speeds up as no obstacle is visible), (4) abruptly slow the vehicle-in-front to trigger a collision (violation of safety requirement).

To generate a test case (i.e., a test sequence) for a single requirement (safety, functional), RL is driven by an objective that must be satisfied while interacting with the environment. The objective is to find any

test case t that satisfies $v_{max}(r, d, E, t) > \epsilon_r$, where $v_{max}(r, d, E, t)$ is the maximum degree of violation for a requirement r observed over the simulation of the DADS under test d in its driving environment E, while executing t and assuming ϵ_r is the threshold specifying the maximum acceptable violation for r. The goal of RL-based testing is to find a sequence of changes in the environment that results in satisfying the objective; these changes are stored in t in the form of state-action pairs, where a state captures a snapshot of E and the ego vehicle and an action indicates the change to be applied to E given the state. Storing states in t is essential as it provides necessary information for explaining the changes in the environment that resulted in a requirement violation. A single test case is therefore composed of a sequence of state-action pairs leading to requirement violations.

As described in background chapter (section 2.4), RL methods can be categorised into two types: (1) tabular-based and (2) approximation-based. Considering the simplicity and fast convergence of tabular-based methods, we use Q-learning [136], one of the most widely used algorithms in this category, as our basis in the rest of the chapter. Nevertheless, one can easily opt for other tabular-based RL methods, such as SARSA [108], by just changing the way of updating the Q-table.

To use Q-learning for testing, it is essential to define states, actions, and rewards for an RL agent as described in Section 2.4. In the context of DADS testing, *states* can be defined to capture important details of the simulation (e.g., locations/speeds of actors, weather conditions), *actions* are the environment changes (e.g., change in the dynamics of actors and weather conditions) and *rewards* should indicate the degree of requirements violations. The higher the degree of violation, the higher the reward, so that the RL agent can generate a sequence of state-action pairs that maximizes the sum of rewards.

Algorithm 6 presents a generic RL-based testing algorithm that takes as inputs an objective o, an environment E and a Q-table q (possibly initialized based on prior knowledge), and returns a test case t achieving o and a Q-table q that was updated during the generation of t. If the algorithm cannot find a t that satisfies o, it returns a null value for t along with the updated Q-table q resulting from the search, which can be reused later if needed.

The algorithm begins with the loop for finding t that satisfies o. Until the budget (e.g., total number of hours or simulator runs) runs out, the algorithm repeats the following steps: (1) initialize t and resetting E to its initial state (lines 2–3) and (2) run the tabular-based RL algorithm to generate t (i.e., a sequence of environmental changes in the form of state-action pairs) with the aim of satisfying o (lines 4–12; see below). The algorithm ends by returning t if o is satisfied; otherwise, a null value is returned for t.

To generate t so that it satisfies o (lines 4–12), the algorithm repeats the following steps until the stopping condition (e.g., satisfying o or no more possible actions) is met: (i) observe the state s from E (line 5), (ii) choose an action a either randomly (with a small probability ϵ to increase the exploration of the state space and to avoid being stuck in local optima) or using q and s (line 6), (iii) perform a to update E and receive a reward w for o (line 7), (iv) append a new state-action pair (s, a) at the end of t (line 8), (v) update q using s, a and w (line 9), and (vi) return t and q if the objective o is satisfied (i.e., a violation is found) (line 12).

6.2.2 Test Suite Generation using many-objective RL

Algorithm 6 works well with one objective (violating one requirement) while the nature of our problem, as described in Section 6.1, involves multiple independent objectives. Therefore, we need to extend the algorithm above to efficiently take into account many objectives.

Algorithm 6: RL-based Test Generation (single objective)

```
Input  : Objective o,
         Environment E,
         Q-table q
Output : Test Case t
while not(budget_finished) do
    Test Case t ← ∅
    E ← reset(E)
    while not(stopping_condition) do
        State s ← observe(E)
        Action a ← chooseAction(q, s)
        Reward w ← perform(E, a)
        t ← append(t, (s, a))
        q ← updateQtable(q, s, a, w)
        if satisfy(t, o) then
            return t, q
        end
    end
end
return null, q
```

There is existing work on covering many independent objectives in the context of DES/DADS testing [3, 4, 130]. Though they test both static and dynamic elements of the environment, they do not change the dynamic elements during the execution (simulation) of a test case. Existing approaches can be used for the problem of DADS testing with dynamically changing environments if they extend the search space to take into account the environment's dynamic elements over a certain time horizon; however, this would be highly inefficient due to the resulting much larger search spaces (see Section 6.3 for details).

To efficiently solve the problem of DES online testing considering dynamically changing environments, with many independent objectives, we propose a novel approach: Many-Objective Reinforcement Learning for Online Testing (MORLOT). It combines two distinct techniques: (1) *tabular-based Reinforcement Learning (RL)* to dynamically interact with the environment for finding the environmental changes that cause the violation of given requirements and (2) *many-objective search* for test suite generation [96, 4, 130] to achieve many independent objectives (i.e., violating the requirements) individually within a limited time budget.

Similar to existing work, MORLOT uses the notion of archive to keep the minimal set of test cases satisfying the objectives. To take into account many independent objectives simultaneously, we extend Algorithm 6 to have multiple Q-tables, each of them addressing one objective. Intuitively, each Q-table captures the best action to select for one corresponding objective in a given state. However, the challenge is that, in the same states, different actions can be chosen for different objectives (by different Q-tables). To choose a single action to perform, we select the Q-table based on the objective that achieved the maximum fitness value (i.e., reward in RL) in the previous iteration. This is because that objective is the closest to being satisfied.

MORLOT takes a set of objectives O, an environment E and a set of Q-tables Q (possibly initialized based on prior knowledge); MORLOT returns a test suite containing a test case for each satisfied objective. As stated earlier, we define each objective as a violation of a certain requirement. Specifically, given a set

of requirements $R = \langle r_1, r_2, \ldots r_n \rangle$ for the DADS d, we define a set of objectives $O = \langle o_1, o_2, \ldots o_n \rangle$ where o_i is to cause d to violate r_i for $i = 1, 2, \ldots, n$. MORLOT returns a test suite $TS = \langle t_1, t_2, \ldots, t_m \rangle$ where t_l is a test case satisfying any one of the objective $o_i \in O$ and $m \leq n$.

Algorithm 7 shows the pseudocode of MORLOT. It takes O, E and Q as inputs and returns a test suite containing test cases satisfying at least one objective and multiple Q-tables, one for each objective.

Algorithm 7: MORLOT

```
   Input  : Set of Objectives O
            Environment E
            Set of Q-tables Q
   Output : Archive (Test Suite) A
            Set of Q-tables Q
 1 Set of Uncovered Objectives U ← O
 2 Archive A ← ∅
 3 while not(budget_finished) do
 4     Set of Rewards W ← ∅
 5     Test Case t ← ∅
 6     E ← reset(E)
 7     while not(stopping_condition) do
 8         State s ← observe(E)
 9         Action a ← chooseActionMultiObjs(s, R, Q, U)
10         W ← performMultiObjs(a, E)
11         Q ← updateQtables(Q, s, a, W)
12         t ← append(t, (s, a))
13         foreach o ∈ O do
14             if satisfy(t, o) then
15                 A ← updateArchive(A, t, o)
16                 U ← U − {o}
17             end
18         end
19     end
20 end
21 return A, Q
```

The algorithm starts by initializing the set of uncovered objectives U with O (line 1). It is important to keep a record of uncovered objectives so that the search process can focus on them. It then initializes A (line 2). Notice that $|Q| = |O|$ so that there is a Q-table for each objective. Until the search budget runs out, the algorithm repeats the following steps: (1) initialize a set of rewards W and a test case t and resetting E to its initial state (lines 4–6) and (2) find t that satisfies $u \in U$ using RL (lines 7–19). To achieve the latter, the algorithm repeats the following steps until the stopping conditions are met: (i) observe s from E (line 8), (ii) choose an action a either randomly (with a small probability ϵ to increase the exploration of the state space and to avoid being stuck in local optima) or using a Q-table $q_m \in Q$ and s where q_m is the Q-table of an uncovered objective $u \in U$ whose reward $w \in W$ for the previously chosen action is the maximum (line 9), (iii) perform a to update W received from E (line 10), (iv) update Q using s, a, and W (line 11), (v) append (s, a) at the end of t (line 12), and (vi) update A and U, if t satisfies any $o_i \in O$, such that A includes the shortest test case satisfying o_i from $A \cup \{t\}$ and U excludes o_i (lines 13–16). The algorithm ends by returning A (i.e., a minimal set of test cases, each of them

covering at least one objective) and Q (i.e., a set of filled Q-tables, each of them matching one objective).

Notice that MORLOT updates the Q-tables Q even for covered objectives, while addressing the uncovered objectives, as Q can be reused later for a newer version of the DES under test in a regression testing setting. Since the Q-tables record the best actions to choose for given states, using them for testing the newer versions of the DES can boost the performance of Algorithm 6. This investigation is however left to future work.

6.3 Evaluation

This section reports on the empirical evaluation of MORLOT when testing an open-source DADS. Specifically, we answer the following research questions:

RQ1: How does MORLOT fare compared to other many-objective search approaches tailored for test suite generation in terms of *test effectiveness*?

RQ2: How does MORLOT fare compared to other many-objective search approaches tailored for test suite generation in terms of *test efficiency*?

To answer RQ1, we compare test suites generated by different approaches within the same execution time budget (in computing hours) in terms of their ability to reveal safety and functional requirements violations. To answer RQ2, we compare different approaches in terms of the execution time required to reveal a certain number of requirements violations and how differences among them evolve over time. These investigations aim to evaluate the benefits of MORLOT for DADS online testing, in terms of test effectiveness and efficiency, and therefore the benefits of dynamically changing the environment based on the simulation state.

6.3.1 Evaluation Subjects

We use TransFuser (TF) [102], the highest rank DADS among publicly available ones in the CARLA Autonomous Driving Leaderboard Sensors Track [124] at the time of our evaluation. The Leaderboard evaluates the driving performance of ADS in terms of 11 different metrics designed to assess driving safety, such as red light infractions, collision infractions, and route completion. The driving performance results of TF reported in the Leaderboard show that it is well-trained and able to pass a large variety test scenarios. It ought therefore to be representative of what one can find in the industry.

TF takes an image from the front-facing camera and the sensor data from LiDAR as input and generates the driving command (steering, throttle, and braking). Internally, it uses ResNet34 and ResNet18 [54] to extract features from the input image and sensor data, respectively. It then uses transformers [133] to integrate the extracted image and LiDAR features. The integrated features are processed by a way-point prediction network that predicts the ego vehicle's expected trajectory, which is used for determining the driving command for next time steps.

We also use CARLA [30], a high-fidelity open-source simulator developed for autonomous driving research. CARLA provides hand-crafted static and dynamic elements for driving simulations. Static elements include different types of roads, buildings, and traffic signs. Dynamic elements include other vehicles, pedestrians, weather, and lighting conditions. In our evaluation, we let the approach under evaluation (i.e., MORLOT and its alternatives) control a subset of dynamic elements to mimic real-world

scenarios, such as weather and lighting conditions and the behavior of pedestrians, which are dynamically controllable during the simulation. They also control the throttle and steering of the Vehicle-In-Front (VIF), which is one of the most influential factors in the driving performance of the Ego Vehicle (EV). Furthermore, to avoid trivial violations of safety and functional requirements resulting from the behavior of dynamic elements (e.g., a pedestrian runs into the EV), we manually imposed constraints on such behaviors. The details of the constraints can be found in the supporting material (see Section 6.6).

Considering the capability of the simulator, we use the following six safety and functional requirements: r_1: the EV should not go out of lane; r_2: the EV should not collide with other vehicles; r_3: the EV should not collide with pedestrians; r_4: the EV should not collide with static meshes (i.e., traffic lights, traffic signs etc.); r_5: the EV should reach its destination in defined time budget; r_6: the EV should not violate traffic lights.

Recall that we should specify an initial environment that determines the static elements and the initial states of the dynamic elements for simulation. In practice, one can randomize the initial environments to test diverse scenarios. In our evaluation, however, we need the same initial environments for different approaches (and their repeated runs) to compare them fairly in terms of test effectiveness and efficiency. Since the road type defined in the initial environment is one of the critical factors that has the greatest influence on the driving performance of a DADS, we consider three different initial environments having three different road types: *Straight*, *Left-Turn*, and *Right-Turn*. For the other environmental elements, we use the basic configuration (i.e., sunny weather, the VIF is 10 meters away from the EV, pedestrians are 20 meters away from the EV on a footpath, zero precipitation deposit on roads) provided in CARLA [30]. The details of the initial environment setup can be found in the supporting material (see Section 6.6).

Due to the execution time of individual simulations in CARLA (i.e., 5 minutes on average), the total computing time for all the three different initial environments is more than 600 hours (25 days). To address this issue, we conduct our evaluation on two platforms, P1 and P2. Platform P1 is a desktop with Intel i9-9900K CPU, RTX 2080 Ti (11 GB) GPU, and 32 GB memory, running Ubuntu 18.04. Platform P2 is a `g4dn.xlarge` node configured as Deep Learning AMI (version 61.1) in Amazon Elastic Cloud (`https://aws.amazon.com/ec2/`) with four virtual cores, NVIDIA T4 GPU (16GB), and 16 GB memory, running Ubuntu 18.04. Specifically, we use P1 for the *Straight* environment and five instances of P2 for the remaining. By doing this, we can compare the results of different approaches (i.e., MORLOT and its alternatives) for the same initial environment.

6.3.2 RQ1: Test Effectiveness

Setup

To answer RQ1, we generate test suites using MORLOT and other many-objective search approaches tailored for test suite generation using the same execution time budget. We compare the approaches in terms of *Test Suite Effectiveness (TSE)*. Specifically, the *TSE* of a test suite TS is defined as the proportion of requirements TS violated over the total number of requirements (i.e., six as explained in Section 6.3.1).

To use MORLOT, we need to define states, actions, and rewards specific to our case study as we rely on a tabular-based RL method as mentioned in Section 6.2.1. In the context of DADS online testing, a state should contain all the information that may affect the requirements violations of the DADS, such as weather conditions and the dynamics of the Ego Vehicle (EV), Vehicle-In-Front (VIF), and pedestrian in terms of positions, speeds, and accelerations. To reduce the state space, we consider only one VIF and

one pedestrian since they are sufficient to generate critical test scenarios. Further, we rely on the spatial grid [73] and divide the road into 10x10 grids when representing the positions of the EV and VIF. For speed and acceleration, we use values reported by CARLA, rounded to one decimal point. Specifically, we define a state s as a 6-tuple $s = (EV, VIF, P, h, f, g)$ where each of its elements is defined as follows:

- $EV = (x^{EV}, y^{EV}, v_x^{EV}, v_y^{EV}, a_x^{EV}, a_y^{EV})$ is the state of the EV where x^{EV} and y^{EV} are the x and y components of the absolute position of the EV on the road, v_x^{EV} and v_y^{EV} are the x and y components of the absolute speed of the EV, and a_x^{EV} and a_y^{EV} are the x and y components of the absolute acceleration of the EV.

- $VIF = (x^{VIF}, y^{VIF}, v_x^{VIF}, v_y^{VIF}, a_x^{VIF}, a_y^{VIF})$ is the state of the VIF where x^{VIF} and y^{VIF} are the x and y components of the position of the VIF relative to the EV, v_x^{VIF} and v_y^{VIF} are the x and y components of the speed of the VIF relative to the EV, and a_x^{VIF} and a_y^{VIF} are the x and y components of the acceleration of the VIF relative to the EV.

- $P = (x^P, y^P, v_x^P, v_y^P)$ is the state of the pedestrian where x^P and y^P are the x and y components of the direction of the pedestrian and v_x^P and v_y^P are the x and y components of the speed of the pedestrian. We do not consider the acceleration of the pedestrian since it is not computed in CARLA.

- h is the weather state and can have a value ranging between 0 (clear weather) and 100 (thunderstorm) in steps of 2.5.

- f is the fog state and can have a value ranging between 0 (no fog) and 100 (heavy fog) in steps of 2.5.

- g is the lighting state (controlled by changing the location of the light source) and can have a value ranging between -30 (night) and 120 (evening) in steps of 2.5.

For actions, we keep the size of unit changes of the dynamic elements small, based on preliminary experiments, to avoid unrealistic changes within each time step; for example, the VIF's throttle can increase/decrease by only 0.1 at each simulation time step. As a result, for each time step, one of the following actions can be taken:

- increasing/decreasing throttle by 0.1 (throttle range: 0–1),

- increasing/decreasing steering 0.01 (steering range: -1–1),

- increasing/decreasing light intensity by moving the source of light by 2.5 degrees,

- increasing/decreasing weather intensity by 2.5,

- increasing/decreasing fog intensity by 2.5,

- increasing/decreasing pedestrian speed by 0.05 m/s (speed range 0.3–1.5),

- changing pedestrian direction (x,y-axis) by 0.1 (direction change range: -1–1), and

- do nothing.

For rewards, we define one reward function for each requirement r_i for $i = 1, \ldots, 6$, discussed in Section 6.3.1. Since we aim to cause the DADS to violate the requirements, we need higher reward values for more critical situations. For $i = 1, \ldots, 5$, the criticality of a situation depends on the distance between the EV and the object (i.e., the VIF, pedestrian, static meshes, center of the lane, and destination for r_1, r_2, r_3, r_4, and r_5, respectively); the shorter the distance, the more critical. To capture this, we define the reward function $reward_{1,\ldots,5}$ for $r_1, \ldots, r_5$ as follows:

$$reward_{1,\ldots,5} = \begin{cases} 1/d_{EV,obj}, & \text{if } d_{EV,obj} > 0 \\ 1000000, & \text{else} \end{cases}$$

where $d_{EV,obj}$ refers to the distance between the EV and the object and 1,000,000 is the maximum reward for any violation, a large number which is not achievable without violations. The distance is normalized between 0 and 1 so that every requirement contributes equally to the reward function. For r_6, $reward_6$ is defined as binary due to the limitation of CARLA: it returns 1 if r_6 is violated (i.e., at least one traffic rule is violated); otherwise 0.

Besides the definition of states, actions, and rewards for MORLOT, there are a few RL parameters to be tuned [121], such as the probability ϵ of choosing a random action, the learning rate α, and the discount factor γ. For ϵ, to make MORLOT more exploratory at first but gradually more exploitative, we dynamically decrease it from 1.0 to 0.1 during the search, following a suggested approach [89]. Specifically, we decrease it from 1 to 0.1 in the first 20% of the search budget and keep it at 0.1 for the rest. For α and γ, based on preliminary experiments, we set the values to 0.01 and 0.9, respectively.

For comparison with MORLOT, we use two well-known approaches for many-objective test suite generation: MOSA [96] and FITEST [4]. The six reward functions defined above for MORLOT are used as fitness functions for MOSA and FITEST. Recall that, as described in Section 6.1, our test case (test scenario) is a sequence of values of the dynamic elements for J time steps, where the length of a simulation J can vary depending on the behavior of dynamic elements. Since MOSA and FITEST search for test cases that satisfy many test objectives without sequentially determining environmental changes, the length of a test case (i.e., the length of a sequence of values of the dynamic elements) should be fixed before running the search. Therefore, we set the length of a test case as the maximum possible length of a simulation determined by the minimum car speed and the maximum road length; if a simulation ends before its maximum possible length, then the remaining sequence in a test case is ignored. To be consistent with previous studies [130, 4], we set the population size equal to the number of objectives (i.e., requirements). We follow the original studies [4, 96] for mutation and crossover rates.

We also use Random Search (RS) with an archive as a baseline. RS generates random changes to the dynamic elements as a test case. The possible changes are the same as those of MORLOT, MOSA, and FITEST. Furthermore, RS maintains an archive for all the test cases that lead to requirements violations. RS will provide insights on how complex the search problem is and will help us quantify the relative effectiveness of advanced approaches: MORLOT, MOSA, and FITEST.

Note that we do not additionally consider other RL-based testing approaches for comparison since they do not account for cases where many requirements must be validated at the same time. As explained in Section 6.1, simply repeating a single-objective approach multiple times, by dividing the test budget per requirement, cannot scale in our context, given that test executions are expensive due to high-fidelity simulations.

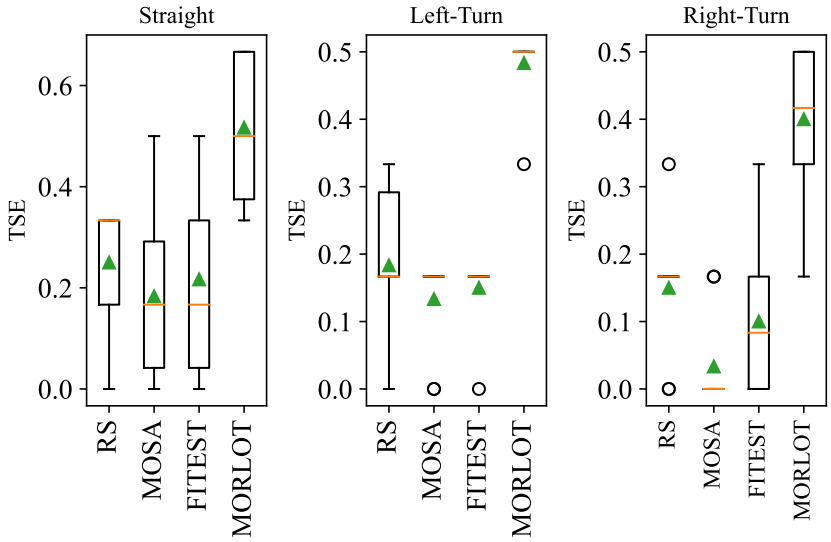


Figure 6.1: Distribution of *TSE* values for different testing approaches

Table 6.1: Statistical comparison results for different approaches

Comparison		*Straight*		*Left-Turn*		*Right-Turn*	
A	B	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$
FITEST	RS	0.605	0.43	0.261	0.42	0.269	0.37
MOSA	RS	0.261	0.35	0.120	0.38	0.009	0.19
MORLOT	RS	0.001	0.91	0.000	0.99	0.001	0.94
MOSA	FITEST	0.753	0.46	0.394	0.45	0.161	0.34
MORLOT	FITEST	0.003	0.88	0.000	1.00	0.001	0.95
MORLOT	MOSA	0.001	0.93	0.000	1.00	0.000	0.99

To account for randomness in all the approaches, we repeat the experiment 10 times with the same time budget of four hours. We found that fitness reaches a plateau after four hours based on our preliminary evaluations. We apply Mann–Whitney U tests [83] to evaluate the statistical significance of differences in *TSE* values among the approaches. We also measure Vargha and Delaney's $\hat{A}_{AB}$ [132] to calculate the effect size of the differences; $\hat{A}_{AB}$ ($= 1 - \hat{A}_{BA}$) indicates that A is better than B with a small, medium, and large effect size when its value exceeds 0.56, 0.64, and 0.71, respectively.

Results

Figure 6.1 shows the distribution of *TSE* values achieved by RS, MOSA, FITEST, and MORLOT over 10 runs for each of the three initial environments (i.e., *Straight*, *Left-Turn*, and *Right-Turn*). The orange bar and green triangle in the center of each box represent the median and average, respectively. In addition, Table 6.1 shows the statistical comparison results between different approaches. The columns A and B indicate the two approaches being compared. The columns p-value and $\hat{A}_{AB}$ indicate the statistical significance and effect size, respectively, when comparing A and B in terms of *TSE*.

For all three initial environments, it is clear that MORLOT outperforms RS, MOSA, and FITEST. Given a significance level of $\alpha = 0.01$, the differences between MOTLOT and the others are significant (p-value < 0.01) in all cases. Furthermore, $\hat{A}_{AB}$ is always greater than 0.71 when $A =$ MORLOT, meaning that MORLOT has a large effect size when compared to the others in terms of *TSE*. This result implies that, by combining RL and many-objective search, MORLOT can detect significantly more violations for a given set of safety and functional requirements than random search and state-of-the-art many-objective search approaches for test suite generation. This is mainly because MORLOT can incrementally generate a sequence of changes by observing the state and reward after each change, whereas the other approaches must generate an entire sequence at once whose fitness score is calculated only after the simulation is completed.

It is also interesting to see that MOSA and FITEST do not outperform RS in all cases. Given $\alpha = 0.01$, the differences between MOSA and RS and between FITEST and RS are insignificant, except for the difference between MOSA and RS in the *Right-Turn* environment. This means that, in the *Straight* and *Left-Turn* environments, the advanced many-objective approaches are not significantly better than simple random search with an archive. One possible explanation can be that the search space considering dynamically changing environmental elements is enormously large, making advanced search approaches less effective within the given time budget (i.e., four hours). Although they might perform better than random search if given much more time, this is unrealistic in practical conditions.

> The answer to RQ1 is that MORLOT is significantly more effective, in terms of *Test Suite Effectiveness (TSE)*, and with a large effect size, than random search and alternative many-objective search approaches tailored for test suite generation.

6.3.3 RQ2: Test Efficiency

Setup

To answer RQ2, we basically use the same setup as in RQ1 but additionally measure the achieved *TSE* values at 20-minute intervals over a 4-hour run. To account for randomness, again, we repeat the experiment 10 times and report on how the average *TSE* values for 10 runs vary over time from 20 minutes to 240 minutes in steps of 20 minutes.

Results

Figure 6.2 shows the relationship between the execution time and the average *TSE* values for 10 runs of RS, MOSA, FITEST, and MORLOT for each of the three initial environments.

Overall, we can clearly see that MORLOT is always at the top in all environments, meaning that MORLOT always achieves the highest *TSE* values, at any time, when compared to the other approaches over the same period of time. Furthermore, the gaps between MORLOT and the others keep increasing over time, up to a certain point, implying that MORLOT is not only significantly more efficient at detecting unknown violations but also that the effect size becomes larger as we run the testing approaches longer. An tentative explanation for this observation is that MORLOT keeps learning over time as it observes further states and rewards during the generation of test scenarios. Furthermore, as described in Section 6.3.2,

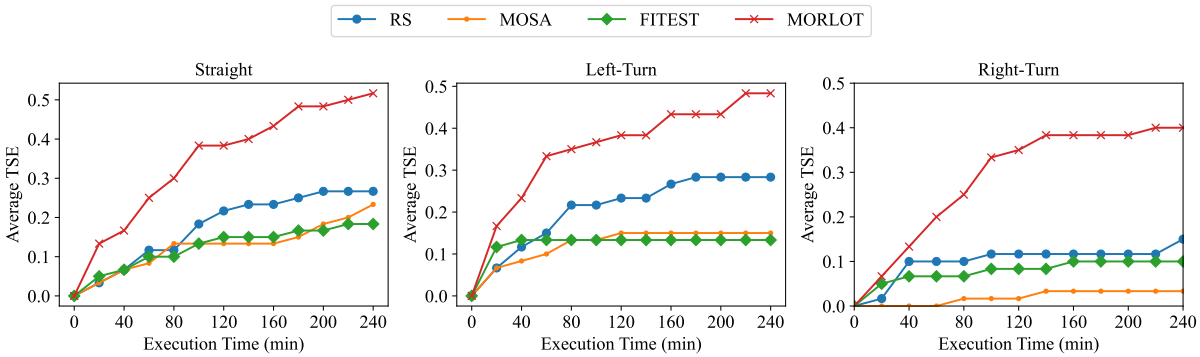


Figure 6.2: Average *TSE* values over 20 minutes interval

dynamically decreasing the epsilon value in MORLOT makes it gradually more exploitative and more effective.

Comparing RS, MOSA, and FITEST, we can see that MOSA and FITEST do not significantly outperform RS at any time, meaning that advanced search-based approaches are not more efficient than simple random search with an archive in this context. As already discussed in RQ1, this can be mainly because of the enormous search space that makes advanced search approaches less effective within a practical time budget.

> The answer to RQ2 is that MORLOT is significantly more efficient than random search and alternative many-objective search approaches. Indeed, it achieves, for any given time budget, a significantly higher average *TSE*, and this difference keeps increasing over time.

6.3.4 Threats to Validity

Using one DADS and one simulator is a potential threat to the external validity of our results. To mitigate the issue, we selected the highest rank DADS (i.e., Transfuser), at the time of our evaluation, among publicly available ones in the CARLA Automation Driving Leaderboard [124], and a high-fidelity driving simulator (i.e., CARLA) that can be coupled with the selected DADS; they are representative of state-of-the-art DADS and advanced driving simulators, respectively, in terms of performance and fidelity [30, 102]. Note that we did not consider more DADS from the Leaderboard since (1) most of them are not good enough to drive the ego vehicle safely (e.g., yielding many violations of the given requirements even by random search) and (2) our evaluation already took more than 600 computing hours. Nevertheless, further studies will be needed to increase the generalizability of our results.

The degree of the discretization of the actions and states could be a potential factor that affects our results. However, the same set of possible actions (changes) is used for RS, MOSA, FITEST, and MORLOT. Furthermore, since only MORLOT considers states, using a sub-optimal discretization of these states would only decrease the effectiveness and efficiency of MORLOT, and therefore this aspect has not impact on our conclusions.

6.4 Related Work

This section discusses DES testing in terms of two approaches closely related to ours: search-based testing and RL-based testing.

6.4.1 Search-Based Testing

Search-based testing has been widely adapted to test DES, particularly DADS, by formulating a test data generation problem as an optimization problem where an objective is to cause the DADS under test to misbehave and applying metaheuristic search algorithms to solve the optimization problem automatically. For example, Gambi et al. [41] presented ASFAULT, a tool for generating test scenarios in the form of road networks using a genetic algorithm to cause the DADS under test to go out of lane. Klischat and Althoff [65] tested motion-planning modules in DADS by generating critical test scenarios based on a minimization of the search space, defined as a set of scenario parameter intervals, of the vehicle under test. Riccio and Tonella [104] presented DEEPJANUS, an approach that uses NSGA-II to generate a *pair* of close scenarios, where the DADS under test misbehaves for one scenario but not for the other, in an attempt to find a *frontier* behavior at which the system under test starts to misbehave. For a configurable, parameterized ADS, Calò et al. [17] aimed to find *avoidable* collisions, that is collisions that would not have occurred with differently-configured ADS; using NSGA-II, they first search for a collision scenario and then search for a new configuration of the ADS which avoids the collision. Considering the high computational cost of simulations involved in search-based approaches, improving the efficiency of search-based testing has also been studied in the context of DES/DADS testing. Abdessalem et al. [3] presented NSGAII-DT that combines NSGA-II (a multi-objective search algorithm) and Decision Tree (a classification model) to generate critical test cases while refining and focusing on the regions of a test scenario space that are likely to contain cost critical test scenarios. Ul Haq et al. [130] presented SAMOTA, an efficient online testing approach extending many-objective search algorithms tailored for test suite generation to utilize *surrogate models* that can mimic driving simulation (e.g., whether a requirement violation occurred or not) and are much less expensive to run.

Though search-based testing approaches have shown to work very well for testing DADS when the environment remains static throughout the tested scenario, they are not likely to work efficiently in the case of dynamically changing environments. One of the reasons is they must get the fitness score for a test scenario after the simulation is completed (when we can ascertain whether a requirement violation occurred) and cannot, therefore, change the environment during a simulation run. Another challenge is that the search space becomes enormous because of the many possible combinations of environment parameters at each timestamp.

6.4.2 RL-Based Testing

RL has also recently been used for DES/DADS testing. In RL-based testing, a DES/DADS testing problem is formulated as a sequential decision-making problem where the goal is to generate a compact test scenario (in the form of a sequence of environmental changes) that causes the system under test to violate a given requirement. For example, Koren et al. [69] presented an approach that extends Adaptive Stress Testing (AST) [72] by using reinforcement learning for updating the environment of a vehicle to cause a collision. Corso et al. [26] further extended AST to include domain relevant information in

the search process. This modification helps in finding a more diverse set of test scenarios in the context of DADS testing. Sharif and Marijan [113] presented a two-step approach that not only generates test scenarios using Deep RL but also utilizes the test scenarios to improve the robustness of the DADS under test by retraining it. Very recently, Lu et al. [80] presented DEEPCOLLISION, an approach that learns the configurations of the environment, using Deep Q-learning, that can cause the crash of the ego vehicle.

Despite encouraging achievements when using RL for DES/DADS testing, especially when the objective is to dynamically change the environment, there is no work that focuses on testing many independent requirements simultaneously, thus raising the problems discussed in Section 6.1.

6.5 Conclusion

In this chapter, we present MORLOT, a novel approach that combines Reinforcement Learning (RL) and many-objective search to effectively and efficiently generate a test suite for DNN-Enabled Systems (DES) by dynamically changing the application environment. We specifically address the issue of scalability when many requirements must be validated. We empirically evaluate MORLOT using a state-of-the-art DNN-enabled Automated Driving System (DADS) integrated with a high-fidelity driving simulator. The evaluation results show that MORLOT is significantly more effective and efficient, with a large effect size, than random search and many-objective search approaches tailored for test suite generation.

6.6 Data Availability

The replication package of our experiments, including the implementation of MORLOT and alternative approaches, the instructions to set up and configure the DADS and simulator, and the detailed descriptions of the initial environments used in the experiments, is currently under review for its open-source license by our legal team.

Chapter 7

Conclusion & Future Work

In this chapter, we summarise the contributions presented in this dissertation and discuss potential future work.

7.1 Summary

In this dissertation, we focused on providing scalable and practical solutions for testing deep neural networks (DNNs) and DNN-Enabled Systems (DES). The main challenges for testing DNNs include: (1) the extremely large search space for all possible test data, (2) the many requirements to be considered at the same time, (3) computationally expensive simulation, and (4) no access to internal information of DNNs.

Throughout this dissertation, we propose several approaches to address the aforementioned challenges in collaboration with IEE Sensing, who provides advanced sensing solutions to the automotive industry. Specifically, the dissertation covered the following in each chapter.

Chapter 3 presented an empirical study to compare offline and online testing in the context of autonomous driving systems. We investigated the use of simulator-generated data in lieu of real-world data. We also investigated whether offline testing could be used to reduce the cost of online testing. In this empirical study, we used open-source DNNs that predict the steering angles based on an input driving scene (image).

Chapter 4 presented an automated approach to generate test data for key-points detection DNNs using many-objective optimisation. Our approach uses many-objective search algorithms tailored for test suite generation to generate test data. We also demonstrated a way to learn the conditions, in terms of image characteristics, that cause the misprediction for each keypoint. We evaluated our approach using a facial key-point detection DNN developed by IEE. The results showed that our approach can generate test suites to severely mispredict more than 93% of all keypoints on average, while random search, as a comparison, can do so for 41% of them.

Chapter 5 presented SAMOTA, a surrogate-assisted many-objective testing approach. SAMOTA leverages the combination of surrogate-assisted optimisation and many-objective optimisation. SAMOTA

iteratively uses two search phases: global search (to search the complete search space) and local search (to search a promising sub-portion of the search space). Both search phases use surrogate models that are trained with data from the targeted search space. We evaluated the effectiveness and efficiency of SAMOTA using an open source DNN-enabled autonomous driving system, Pylot, integrated with a high-fidelity simulator, CARLA. The results showed that SAMOTA is statistically more effective and efficient than other many-objective search algorithms tailored for test suite generation and random search.

Chapter 6 presented MORLOT, a many-objective reinforcement learning approach for online testing. MORLOT relies on the combination of reinforcement learning, to dynamically interact with the environment, and many-objective optimisation, to solve multiple objectives simultaneously. We evaluated MORLOT using an open source DNN-enabled autonomous driving system, Transfuser, integrated with a high-fidelity simulator, CARLA. The results showed that MORLOT is statistically more effective and efficient than alternative approaches.

7.2 Future Work

Overall, we plan to increase the generalisability of the approaches presented in this dissertation. Specifically, we intend to include recently released end-to-end DNN-enabled autonomous driving systems (DADS) in the evaluation of MORLOT and SAMOTA. We also intend to include more Keypoints detection DNNs in evaluating our approach for test data generation.

For SAMOTA, we plan to investigate further how to explain the safety violations detected by our approach; for example, we can derive association rules between the test scenario attribute values and the resulting safety violations using association rule mining algorithms [55].

For MORLOT, we plan to investigate if we can reuse MORLOT's Q-tables trained on a former version of the DES under test to improve test effectiveness and efficiency for latter versions. We also plan to extend MORLOT to leverage different RL methods, including Deep Q-learning with varying epsilon values, and compare the results in the context of DADS online testing.

We also plan to create a publicly available framework for testing DNN-based systems. We intend to combine all the approaches into one framework where users can test their systems based on four inputs: system or model, system's parameters, simulator, simulator's parameters. The framework would learn and decide which approach to use, starting with determining the type of testing to use, online or offline testing. The framework would start with basic approaches such as many-objective search for test suite generation (Chapter 4) and then move on to more sophisticated approaches such as SAMOTA (Chapter 5) and MORLOT (Chapter 6). At the end of the testing budget, the framework would return a report containing all the details about the types of testing that were carried out and their results.

Bibliography

[1] 2022. Intelligent Sensing Solutions. https://iee-sensing.com/

[2] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 1016–1026. https://doi.org/10.1145/3180155.3180160

[3] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2018. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 1016–1026.

[4] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Autonomous Cars for Feature Interaction Failures Using Many-Objective Search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/3238147.3238192

[5] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Autonomous Cars for Feature Interaction Failures Using Many-Objective Search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/3238147.3238192

[6] Mykhaylo Andriluka, Leonid Pishchulin, Peter V. Gehler, and Bernt Schiele. 2014. 2D Human Pose Estimation: New Benchmark and State of the Art Analysis. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, Columbus, OH, 3686–3693. https://doi.org/10.1109/CVPR.2014.471

[7] Kellie J. Archer and Ryan V. Kimes. 2008. Empirical characterization of random forest variable importance measures. *Computational Statistics and Data Analysis* 52, 4 (2008), 2249 – 2260. https://doi.org/10.1016/j.csda.2007.08.015

[8] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38. https://doi.org/10.1109/MSP.2017.2743240

[9] Team Autumn. 2016. Autumn model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/autumn. Accessed: 2019-10-11.

[10] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785

[11] Halil Beglerovic, Michael Stolz, and Martin Horn. 2017. Testing of autonomous vehicles using surrogate models and stochastic optimization. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 1–6.

[12] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2016. Testing Advanced Driver Assistance Systems Using Multi-Objective Search and Neural Networks. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE 2016)*. Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/2970276.2970311

[13] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*. IEEE, 3464–3468.

[14] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.

[15] Leyde Briceno and Gunther Paul. 2019. MakeHuman: A Review of the Modelling Framework. In *Proceedings of the 20th Congress of the International Ergonomics Association (IEA 2018)*, Sebastiano Bagnara, Riccardo Tartaglia, Sara Albolino, Thomas Alexander, and Yushi Fujita (Eds.). Springer International Publishing, Cham, 224–232.

[16] David S Broomhead and David Lowe. 1988. *Radial basis functions, multi-variable functional interpolation and adaptive networks*. Technical Report. Royal Signals and Radar Establishment Malvern (United Kingdom).

[17] Alessandro Calò, Paolo Arcaini, Shaukat Ali, Florian Hauer, and Fuyuki Ishikawa. 2020. Generating Avoidable Collision Scenarios for Testing Autonomous Driving Systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. 375–386. https://doi.org/10.1109/ICST46399.2020.00045

[18] Team Chauffeur. 2016. Chauffeur model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur. Accessed: 2019-10-11.

[19] Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, and T.H. Tse. 2010. Adaptive Random Testing: The ART of test case diversity. *Journal of Systems and Software* 83, 1 (2010), 60–66. https://doi.org/10.1016/j.jss.2009.02.022 SI: Top Scholars.

[20] Yu Chen, Jian Yang, and Jianjun Qian. 2017. Recurrent neural network for facial landmark detection. *Neurocomputing* 219 (2017), 26–38.

[21] Guillermo Campos Ciro, Frédéric Dugardin, Farouk Yalaoui, and Russell Kelly. 2016. A NSGA-II and NSGA-III comparison for solving an open shop scheduling problem with resource constraints. *IFAC-PapersOnLine* 49, 12 (2016), 1272–1277.

[22] Felipe Codevilla, Antonio M. Lopez, Vladlen Koltun, and Alexey Dosovitskiy. 2018. On Offline Evaluation of Vision-based Driving Models. In *The European Conference on Computer Vision (ECCV)*.

[23] William W. Cohen. 1995. Fast Effective Rule Induction. In *Machine Learning Proceedings 1995*, Armand Prieditis and Stuart Russell (Eds.). Morgan Kaufmann, San Francisco (CA), 115 – 123. https://doi.org/10.1016/B978-1-55860-377-6.50023-2

[24] Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation. http://www.blender.org

[25] Makehumancommunity Online Community. 2020. *Makehumancommunity.org*. Makehumancommunity. www.Makehumancommunity.org

[26] Anthony Corso, Peter Du, Katherine Driggs-Campbell, and Mykel J Kochenderfer. 2019. Adaptive stress testing with reward augmentation for autonomous vehicle validatio. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 163–168.

[27] George E Dahl, Dong Yu, Li Deng, and Alex Acero. 2011. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing* 20, 1 (2011), 30–42.

[28] Kalyanmoy Deb and Himanshu Jain. 2013. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation* 18, 4 (2013), 577–601.

[29] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[30] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 78)*, Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (Eds.). PMLR, 1–16. https://proceedings.mlr.press/v78/dosovitskiy17a.html

[31] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

[32] Tommaso Dreossi, Shromona Ghosh, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2017. Systematic Testing of Convolutional Neural Networks for Autonomous Driving. arXiv:1708.03309 [cs.CV]

[33] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Jianjun Zhao, and Yang Liu. 2018. DeepCruiser: Automated Guided Testing for Stateful Deep Learning Systems. *CoRR* abs/1812.05339 (2018). arXiv:1812.05339 http://arxiv.org/abs/1812.05339

[34] Epic Games. 2019. Unreal Engine. https://www.unrealengine.com

[35] ESI Group. 2019. ESI Pro-SiVIC - 3D Simulations Of Environments And Sensors. https://www.esi-group.com/software-solutions/virtual-environment/virtual-systems-controls/esi-pro-sivictm-3d-simulations-environments-and-sensors. Accessed: 2019-10-11.

[36] P.J Fleming and R.C Purshouse. 2002. Evolutionary algorithms in control systems engineering: a survey. *Control Engineering Practice* 10, 11 (2002), 1223–1241. https://doi.org/10.1016/S0967-0661(02)00081-3

[37] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: Automatic Test Suite Generation for Object-Oriented Software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 416–419. https://doi.org/10.1145/2025113.2025179

[38] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Transactions on Software Engineering* 39, 2 (2013), 276–291. https://doi.org/10.1109/TSE.2012.14

[39] Alessio Gambi. 2012. *Kriging-based self-adaptive controllers for the cloud*. Ph.D. Dissertation. Università della Svizzera italiana.

[40] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically Testing Self-driving Cars with Search-based Procedural Content Generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) *(ISSTA 2019)*. ACM, New York, NY, USA, 318–328. https://doi.org/10.1145/3293882.3330566

[41] Alessio Gambi, Marc Müller, and Gordon Fraser. 2019. AsFault: Testing Self-driving Car Software Using Search-based Procedural Content Generation. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, Piscataway, NJ, USA, 27–30. https://doi.org/10.1109/ICSE-Companion.2019.00030

[42] Alessio Gambi, Giovanni Toffetti, Cesare Pautasso, and Mauro Pezze. 2012. Kriging controllers for cloud applications. *IEEE Internet Computing* 17, 4 (2012), 40–47.

[43] A. Geiger, P. Lenz, and R. Urtasun. 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 3354–3361. https://doi.org/10.1109/CVPR.2012.6248074

[44] Robin Genuer, Jean-Michel Poggi, and Christine Tuleau-Malot. 2010. Variable selection using random forests. *Pattern Recognition Letters* 31, 14 (2010), 2225 – 2236. https://doi.org/10.1016/j.patrec.2010.03.014

[45] Tushar Goel, Raphael T Haftka, Wei Shyy, and Nestor V Queipo. 2007. Ensemble of surrogates. *Structural and Multidisciplinary Optimization* 33, 3 (2007), 199–216. https://doi.org/10.1007/s00158-006-0051-9

[46] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Matthew A. Wright, Joseph E. Gonzalez, and Ion Stoica. 2021. Pylot: A Modular Platform for Exploring Latency-Accuracy Tradeoffs in Autonomous Vehicles. arXiv:2104.07830 [cs.RO]

[47] Object Management Group. 2014. Object Constraint Language Specification. https://www.omg.org/spec/OCL/. Accessed: 2019-10-11.

[48] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: Differential Fuzzing Testing of Deep Learning Systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 739–743. https://doi.org/10.1145/3236024.3264835

[49] Fitash Ul Haq, Donghwan Shin, Lionel C. Briand, Thomas Stifter, and Jun Wang. 2021. Automatic Test Suite Generation for Key-Points Detection DNNs Using Many-Objective Search (Experience Paper). In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, Denmark) *(ISSTA 2021)*. Association for Computing Machinery, New York, NY, USA, 91–102. https://doi.org/10.1145/3460319.3464802

[50] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel Briand. 2020. Supporting materials (temporal link for the double-blind review). http://tiny.cc/Experiment-data. Accessed: 2020-07-26.

[51] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C Briand. 2020. Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 85–95.

[52] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. 2020. Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. 85–95. https://doi.org/10.1109/ICST46399.2020.00019

[53] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. 2021. Can Offline Testing of Deep Neural Networks Replace Their Online Testing? *Empirical Software Engineering* 26, 90 (2021). https://doi.org/10.1007/s10664-021-09982-4

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[55] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. 2000. Algorithms for Association Rule Mining — a General Survey and Comparison. *SIGKDD Explor. Newsl.* 2, 1 (June 2000), 58–64. https://doi.org/10.1145/360402.360421

[56] Yunfei Hou, Yunjie Zhao, Aditya Wagh, Longfei Zhang, Chunming Qiao, Kevin F Hulme, Changxu Wu, Adel W Sadek, and Xuejie Liu. 2015. Simulation-based testing and evaluation tools for transportation cyber–physical systems. *IEEE Transactions on Vehicular Technology* 65, 3 (2015), 1098–1108.

[57] Di Huang, Renke Zhang, Yuan Yin, Yiding Wang, and Yunhong Wang. 2017. Local feature approach to dorsal hand vein recognition by centroid-based circular key-point grid and fine-grained matching. *Image and Vision Computing* 58 (2017), 266–277.

[58] Yichao Huang, Xiaorui Liu, Lianwen Jin, and Xin Zhang. 2015. Deepfinger: A cascade convolutional neuron network approach to finger key point detection in egocentric vision with mobile camera. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, IEEE, Kowloon, 2944–2949.

[59] Rateb Jabbar, Khalifa Al-Khalifa, Mohamed Kharbeche, Wael Alhajyaseen, Mohsen Jafari, and Shan Jiang. 2018. Real-time driver drowsiness detection for android application using deep neural networks techniques. *Procedia computer science* 130 (2018), 400–407.

[60] Ruichen Jin, Wei Chen, and Timothy W Simpson. 2001. Comparative studies of metamodelling techniques under multiple modelling criteria. *Structural and multidisciplinary optimization* 23, 1 (2001), 1–13. https://doi.org/10.1007/s00158-001-0160-4

[61] Yaochu Jin. 2011. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation* 1, 2 (2011), 61–70. https://doi.org/10.1016/j.swevo.2011.05.001

[62] Yaochu Jin and Bernhard Sendhoff. 2009. A systems approach to evolutionary multiobjective structural optimization and beyond. *IEEE Computational Intelligence Magazine* 4, 3 (2009), 62–76. https://doi.org/10.1109/MCI.2009.933094

[63] Nidhi Kalra and Susan M. Paddock. 2016. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice* 94 (2016), 182 – 193. https://doi.org/10.1016/j.tra.2016.09.010

[64] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering*

(Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, Piscataway, NJ, USA, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[65] Moritz Klischat and Matthias Althoff. 2019. Generating Critical Test Scenarios for Automated Vehicles with Evolutionary Algorithms. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2352–2358. https://doi.org/10.1109/IVS.2019.8814230

[66] Joshua Knowles and David Corne. 2007. Quantifying the Effects of Objective Space Dimension in Evolutionary Multiobjective Optimization. In *Evolutionary Multi-Criterion Optimization*, Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 757–771.

[67] Team Komanda. 2016. Komanda model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/komanda. Accessed: 2020-04-14.

[68] Zelun Kong, Junfeng Guo, Ang Li, and Cong Liu. 2020. PhysGAN: Generating Physical-World-Resilient Adversarial Examples for Autonomous Driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, Seattle, WA, USA, 14242–14251. https://doi.org/10.1109/CVPR42600.2020.01426

[69] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. 2018. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1–7.

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[71] Fred Lambert. 2016. Understanding the fatal Tesla accident on Autopilot and the NHTSA PROBE. https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/

[72] Ritchie Lee, Mykel J Kochenderfer, Ole J Mengshoel, Guillaume P Brat, and Michael P Owen. 2015. Adaptive stress testing of airborne collision avoidance systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. IEEE, 6C2–1.

[73] Edouard Leurent. 2018. A survey of state-action representations for autonomous driving. (2018).

[74] J. Li and E. Y. Lam. 2015. Facial expression recognition using deep neural networks. In *2015 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, Macau, 1–6. https://doi.org/10.1109/IST.2015.7294547

[75] Zheng Li, Mark Harman, and Robert M Hierons. 2007. Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering* 33, 4 (2007), 225–237.

[76] Dudy Lim, Yaochu Jin, Yew-Soon Ong, and Bernhard Sendhoff. 2010. Generalizing Surrogate-Assisted Evolutionary Computation. *IEEE Transactions on Evolutionary Computation* 14, 3 (2010), 329–355. https://doi.org/10.1109/TEVC.2009.2027359

[77] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. 2020. Black-box adversarial sample generation based on differential evolution. *Journal of Systems and Software* 170 (2020), 110767.

[78] Qunfeng Liu, Xunfeng Wu, Qiuzhen Lin, Junkai Ji, and Ka-Chun Wong. 2021. A novel surrogate-assisted evolutionary algorithm with an uncertainty grouping based infill criterion. *Swarm and Evolutionary Computation* 60 (2021), 100787. https://doi.org/10.1016/j.swevo.2020.100787

[79] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

[80] Chengjie Lu, Yize Shi, Huihui Zhang, Man Zhang, Tiexin Wang, Tao Yue, and Shaukat Ali. 2022. Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize their Collisions. *IEEE Transactions on Software Engineering* (2022).

[81] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: Multi-granularity Testing Criteria for Deep Learning Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. ACM, New York, NY, USA, 120–131. https://doi.org/10.1145/3238147.3238202

[82] Rupak Majumdar, Aman Mathur, Marcus Pirron, Laura Stegner, and Damien Zufferey. 2019. Paracosm: A Language and Tool for Testing Autonomous Driving Systems. arXiv:1902.01084 [cs.SE]

[83] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[84] Daniel V. McGehee, Elizabeth N. Mazzae, and G.H. Scott Baldwin. 2000. Driver Reaction Time in Crash Avoidance Research: Validation of a Driving Simulator Study on a Test Track. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 44, 20 (2000), 3–320–3–323. https://doi.org/10.1177/154193120004402026 arXiv:https://doi.org/10.1177/154193120004402026

[85] Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software* 2, 11 (2017), 205.

[86] Phil McMinn. 2011. Search-Based Software Testing: Past, Present and Future. In *Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW '11)*. IEEE Computer Society, USA, 153–163. https://doi.org/10.1109/ICSTW.2011.100

[87] Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. 2020. Approximation-Refinement Testing of Compute-Intensive Cyber-Physical Models: An Approach Based on System Identification. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 372–384. https://doi.org/10.1145/3377811.3380370

[88] Ryszard S Michalski. 2000. Learnable evolution model: Evolutionary processes guided by machine learning. *Machine learning* 38, 1 (2000), 9–40. https://doi.org/10.1023/A:1007677805582

[89] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[90] Harvey J. Motulsky and Lennart A. Ransnas. 1987. Fitting curves to data using nonlinear regression: a practical and nonmathematical review. *The FASEB Journal* 1, 5 (1987), 365–374. https://doi.org/10.1096/fasebj.1.5.3315805 arXiv:https://faseb.onlinelibrary.wiley.com/doi/pdf/10.1096/fasebj.1.5.3315805

[91] Galen E Mullins, Paul G Stankiewicz, R Chad Hawthorne, and Satyandra K Gupta. 2018. Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. *Journal of Systems and Software* 137 (2018), 197–215.

[92] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. 2019. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences* 116, 44 (2019), 22071–22080.

[93] Alejandro Newell, Kaiyu Yang, and Jia Deng. 2016. Stacked Hourglass Networks for Human Pose Estimation. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 483–499.

[94] Minxue Pan, An Huang, Guoxin Wang, Tian Zhang, and Xuandong Li. 2020. Reinforcement learning based curiosity-driven testing of Android applications. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 153–164.

[95] A. Panichella, F. M. Kifetew, and P. Tonella. 2015. Reformulating Branch Coverage as a Many-Objective Optimization Problem. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, Graz, Austria, 1–10. https://doi.org/10.1109/ICST.2015.7102604

[96] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2017. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering* 44, 2 (2017), 122–158.

[97] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (Abu Dhabi, United Arab Emirates) *(ASIA CCS '17)*. Association for Computing Machinery, New York, NY, USA, 506–519. https://doi.org/10.1145/3052973.3053009

[98] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. ACM, New York, NY, USA, 1–18. https://doi.org/10.1145/3132747.3132785

[99] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2019. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *GetMobile: Mobile Comp. and Comm.* 22, 3, 36–38. https://doi.org/10.1145/3308755.3308767

[100] Joelle Pineau. 2019. ICSE 2019 Keynote: Building Reproducible, Reusable, and Robust Machine Learning Software. https://2019.icse-conferences.org/details/icse-2019-Plenary-Sessions/20/Building-Reproducible-Reusable-and-Robust-Machine-Learning-Software. Accessed: 2019-10-11.

[101] Dean A Pomerleau. 1989. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*. 305–313.

[102] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. 2021. Multi-Modal Fusion Transformer for End-to-End Autonomous Driving. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[103] Vincenzo Riccio and Paolo Tonella. 2020. *Model-Based Exploration of the Frontier of Behaviours for Deep Learning System Testing*. Association for Computing Machinery, New York, NY, USA, 876–888. https://doi.org/10.1145/3368089.3409730

[104] Vincenzo Riccio and Paolo Tonella. 2020. Model-Based Exploration of the Frontier of Behaviours for Deep Learning System Testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 876–888. https://doi.org/10.1145/3368089.3409730

[105] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. Lgsvl simulator: A high fidelity simulator for autonomous driving. In *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*. IEEE, 1–6.

[106] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim. 2020. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 1–6. https://doi.org/10.1109/ITSC45102.2020.9294422

[107] Andras Rozsa, Manuel Günther, Ethan M Rudd, and Terrance E Boult. 2019. Facial attributes: Accuracy and adversarial robustness. *Pattern Recognition Letters* 124 (2019), 100–108.

[108] Gavin A Rummery and Mahesan Niranjan. 1994. *On-line Q-learning using connectionist systems*. Vol. 37. Citeseer.

[109] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. 2017. Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. arXiv:1708.08296 [cs.AI]

[110] B. Sapp and B. Taskar. 2013. MODEC: Multimodal Decomposable Models for Human Pose Estimation. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Portland, OR, USA, 3674–3681. https://doi.org/10.1109/CVPR.2013.471

[111] John Seymour, Dac-Thanh-Chuong Ho, and Quang-Hung Luu. 2021. An Empirical Testing of Autonomous Vehicle Simulator System for Urban Driving. *arXiv preprint arXiv:2108.07910* (2021).

[112] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2018. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*, Marco Hutter and Roland Siegwart (Eds.). Springer International Publishing, Cham, 621–635.

[113] Aizaz Sharif and Dusica Marijan. 2021. Adversarial Deep Reinforcement Learning for Trustworthy Autonomous Driving Policies. *arXiv preprint arXiv:2112.11937* (2021).

[114] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144. https://doi.org/10.1126/science.aar6404 arXiv:https://www.science.org/doi/pdf/10.1126/science.aar6404

[115] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. 2017. Hand keypoint detection in single images using multiview bootstrapping. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. IEEE, Honolulu, HI, 1145–1153.

[116] Guanglu Song, Yu Liu, Yuhang Zang, Xiaogang Wang, Biao Leng, and Qingsheng Yuan. 2020. KPNet: Towards Minimal Face Detector. arXiv:2003.07543 [cs.CV]

[117] T. Sotiropoulos, H. Waeselynck, J. Guiochet, and F. Ingrand. 2017. Can Robot Navigation Bugs Be Found in Simulation? An Exploratory Study. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 150–159. https://doi.org/10.1109/QRS.2017.25

[118] Michael L Stein. 2012. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media.

[119] Stephen M Stigler. 1974. Gergonne's 1815 paper on the design and analysis of polynomial regression experiments. *Historia Mathematica* 1, 4 (1974), 431–439.

[120] Chaoli Sun, Yaochu Jin, Jianchao Zeng, and Yang Yu. 2015. A two-layer surrogate-assisted particle swarm optimization algorithm. *Soft computing* 19, 6 (2015), 1461–1475. https://doi.org/10.1007/s00500-014-1283-z

[121] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[122] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. 2013. Deep neural networks for object detection. *Advances in neural information processing systems* 26 (2013).

[123] TASS International - Siemens Group. 2019. PreScan: Simulation of ADAS and active safety. https://tass.plm.automation.siemens.com. Accessed: 2019-10-11.

[124] CARLA team. 2022. CARLA Autonomous Driving Leaderboard. https://leaderboard.carla.org/leaderboard/

[125] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 303–314. https://doi.org/10.1145/3180155.3180220

[126] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski. 2018. Simulation-based Adversarial Test Generation for Autonomous Vehicles with Machine Learning Components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. 1555–1562. https://doi.org/10.1109/IVS.2018.8500421

[127] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. 2018. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, IEEE, Changshu, China, 1555–1562.

[128] Udacity. 2016. Udacity Self-Driving Car Challenge 2: Using Deep Learning to Predict Steering Angles. https://github.com/udacity/self-driving-car/tree/master/challenges/challenge-2. Accessed: 2019-10-11.

[129] Udacity. 2016. Udacity self-driving challenge 2, CH2-001 (testing) and CH2-002 (training). https://github.com/udacity/self-driving-car/tree/master/datasets/CH2. Accessed: 2019-10-11.

[130] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient Online Testing for DNN-Enabled Systems using Surrogate-Assisted and Many-Objective Optimization. In *Proceedings of the 44th International Conference on Software Engineering (ICSE'22)*. ACM.

[131] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Replication Package For "Efficient Online Testing for DNN-based Systems using Surrogate-Assisted and Many-Objective Optimization". https://doi.org/10.6084/m9.figshare.16468530

[132] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. https://doi.org/10.3102/10769986025002101 arXiv:https://doi.org/10.3102/10769986025002101

[133] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[134] H. Wang, Y. Jin, and J. Doherty. 2017. Committee-Based Active Learning for Surrogate-Assisted Particle Swarm Optimization of Expensive Problems. *IEEE Transactions on Cybernetics* 47, 9 (2017), 2664–2677. https://doi.org/10.1109/TCYB.2017.2710978

[135] Xinyao Wang, Liefeng Bo, and Fuxin Li. 2019. Adaptive Wing Loss for Robust Face Alignment via Heatmap Regression. Seoul, Korea (South), 6970–6980. https://doi.org/10.1109/ICCV.2019.00707

[136] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[137] Eric W Weisstein. 2004. Bonferroni correction. *https://mathworld. wolfram. com/* (2004).

[138] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85. https://doi.org/10.1007/BF00175354

[139] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-Guided Black-Box Safety Testing of Deep Neural Networks. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dirk Beyer and Marieke Huisman (Eds.). Springer International Publishing, Cham, 408–426.

[140] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-Guided Black-Box Safety Testing of Deep Neural Networks. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dirk Beyer and Marieke Huisman (Eds.). Springer International Publishing, Cham, 408–426.

[141] Ian H Witten and Eibe Frank. 2002. Data mining: practical machine learning tools and techniques with Java implementations. *Acm Sigmod Record* 31, 1 (2002), 76–77.

[142] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*. 3645–3649. https://doi.org/10.1109/ICIP.2017.8296962

[143] L. Wolf, T. Hassner, and I. Maoz. 2011. Face recognition in unconstrained videos with matched background similarity. In *CVPR 2011*. IEEE, IEEE, Providence, RI, 529–534. https://doi.org/10.1109/CVPR.2011.5995566

[144] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) *(ISSTA 2019)*. Association for Computing Machinery, New York, NY, USA, 146–157. https://doi.org/10.1145/3293882.3330579

[145] C. Yang, J. Ding, Y. Jin, and T. Chai. 2020. Offline Data-Driven Multiobjective Optimization: Knowledge Transfer Between Surrogates and Generation of Final Solutions. *IEEE Transactions on Evolutionary Computation* 24, 3 (2020), 409–423. https://doi.org/10.1109/TEVC.2019.2925959

[146] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems* 30, 9 (2019), 2805–2824.

[147] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. 2020. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering* (2020), 1–1.

[148] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. ACM, New York, NY, USA, 132–142. https://doi.org/10.1145/3238147.3238187

[149] Shutong Zhang and Chenyue Meng. 2016. Facial keypoints detection using neural network. *Stanford Report* (2016), 1.

[150] Yan Zheng, Yi Liu, Xiaofei Xie, Yepang Liu, Lei Ma, Jianye Hao, and Yang Liu. 2021. Automatic web testing using curiosity-driven reinforcement learning. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 423–435.

[151] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. 2019. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 772–784.

[152] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. DeepBillboard: Systematic Physical-World Testing of Autonomous Driving Systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 347–358. https://doi.org/10.1145/3377811.3380422

[153] Husheng Zhou, Wei Li, Yuankun Zhu, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2018. DeepBillboard: Systematic Physical-World Testing of Autonomous Driving Systems. arXiv:1812.10812 [cs.CV]

[154] Zongzhao Zhou, Yew Soon Ong, Prasanth B Nair, Andy J Keane, and Kai Yew Lum. 2006. Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37, 1 (2006), 66–76. https://doi.org/10.1109/TSMCC.2005.855506

[155] Zhi Quan Zhou and Liqun Sun. 2019. Metamorphic Testing of Driverless Cars. *Commun. ACM* 62, 3 (Feb. 2019), 61–67. https://doi.org/10.1145/3241979